Relazione Progetto Basi di Dati

Sebastiano Sanson Davide Baggio

Anno Accademico 2022

Contents

1 Abstract

Si vuole realizzare una base di dati per la gestione di una piattaforma di streaming di contenuti musicali e podcast. Tale piattaforma consente agli utenti registrati di usufruire dei servizi messi a disposizione, che variano a seconda del tipo di profilo che variano in "utente", "artista" e "podcaster". L'utente è il mero fruitore dei contenuti pubblicati sulla piattaforma dagli artisti e podcasters, e a seconda del piano di abbonamento selezionato gode di vantaggi specifici. Per potersi iscrivere alla piattaforma deve inserire i propri dati personali, di accesso alla piattaforma e, nel caso in cui abbia selezionato un piano non gratuito, i dati di pagamento. Dall'altra parte ci sono gli artisti e i podcasters, ovvero i creatori dei contenuti pubblicati sulla piattaforma, a disposizione per gli utenti registrati. Tra i due vi è una netta distinzione riguardante ciò che possono pubblicare: i primi trattano unicamente singoli e album musicali, mentre gli ultimi citati si occupano della creazione di podcasts di vario genere.

2 Analisi dei requisiti

2.1 Entità

- **Profilo**: è la generalizzazione delle classi utente, artista e podcaster. Contiene tutte le informazioni basilari per la creazione di un profilo sulla piattaforma e altre appartenenti alla sfera "social".
  - mail
  - nome
  - cognome
  - password
  - followers
- **Utente**: l'utente è specializzazione dell'entità profilo, è colui che usufruisce dei servizi offerti dalla piattaforma. All'utente è concessa la possibilità, a seconda delle proprie esigenze, di selezionare un piano tra i 3 offerti: free, premium.
  - following
  - nickname
- **Artista**: è specializzazione dell'entità profilo, è dedicata unicamente agli artisti e si limita alla pubblicazione di nuovi brani musicali, a patto che abbiano stipulato un contratto con un'etichetta discografica (label).
  - nome
  - info
  - ascolti_mensili
  - label
- **Podcaster**: è specializzazione dell'entità profilo, il podcaster è colui che pubblica nuovi podcast
  - nome
  - info
- **Playlist**: è una feature disponibile solo agli utenti che hanno sottoscritto un piano a pagamento, consente di raggruppare un numero non specificato di canzoni a seconda della propria preferenza. Un utente può creare quante playlist desidera.
  - playlist_id
  - nome
  - data_creazione
  - descrizione

- **Piano**: il piano, indipendentemente dall'abbonamento selezionato, specifica la durata di quest'ultimo, partendo dal periodo minimo di un mese.

  - utente
  - inizio_piano
  - fine_piano

- **Abbonamento**: l'entità abbonamento raggruppa i piani presenti sulla piattaforma, descrivendone il costo mensile e fornendo una spiegazione sui vantaggi che lo stesso offre. Le tipologie di abbonamento attualmente disponibili sono le seguenti:

  1. **FREE**: è un abbonamento gratuito che permette la riproduzione di brani e podcast con interruzioni pubblicitarie e la creazione di playlist
  2. **PREMIUM**: grazie al pagamento di una quota mensile si ha la possibilità di riprodurre i contenuti senza alcuna interruzione pubblicitaria

  - nome
  - descrizione
  - costo_mensile

- **Dati Fatturazione**: il pagamento è gestito mediante i dati di fatturazione, appositamente inseriti dall'utente, che contengono le informazioni necessarie per processare la transazione.

  - fatturazione id
  - codice_fiscale
  - via
  - città
  - civico
  - stato
  - cap

- **Carta di Credito**: contiene le informazioni aggiuntive ai dati di fatturazione, sono necessarie per poter prelevare il denaro dal conto selezionato dall'utente.

  - numero
  - intestatario
  - circuito
  - scadenza
  - cvv

- **Pagamento**: entità che tiene traccia dei pagamenti effettuati dall'utente, ognuno identificato da un codice univoco, l'ammontare viene calcolato in base al periodo di abbonamento e la data in cui avviene la transazione.

  - transazione id
  - importo
  - data_fattura

- **Album**: un album contiene una raccolta di brani pubblicati e selezionati da uno o più artisti e viene identificato da un codice univoco in quanto possono esistere più album con uno stesso nome.

  - album id
  - artista
  - titolo
  - data_pubblicazione

- **Canzone**: una canzone è una entità caratterizzata da un titolo e dalla durata. Nel caso un'artista decidesse di pubblicare un "singolo", viene pubblicato sotto un album con il medesimo titolo e contenente unicamente quella canzone specifica.
  - album
  - titolo
  - durata
- **Podcast**: è l'entità che rappresenta un contenuto audio di varie categorie di natura seriale.
  - podcast id
  - nome_podcast
  - podcaster
  - descrizione
- **Categoria**: è l'entità che indica uno o molteplici generi di un podcast.
  - nome
  - descrizione
- **Episodio**: è l'entità che rappresenta un episodio appartenente ad un podcast.
  - podcast
  - titolo
  - durata
  - descrizione
  - data_pubblicazione

2.2 Relazioni

- **Creazione**: rappresenta le *playlist* create dall'utente.
  - **Cardinalità**: un utente può (opzionalmente) scegliere se creare delle playlist *(0,n)*, mentre una playlist può essere create da un solo utente *(1,1)*.
- **Contenuto Playlist**: indica le *canzoni* contenute in una *playlist*.
  - **Cardinalità**: una playlist deve contenere almeno una canzone *(0,n)*, mentre una canzone può essere presente oppure no in una playlist *(0,m)*.
- **Pubblicazione musica**: associa ad uno o più *artisti* gli album o le *canzoni* da essi pubblicati.
  - **Cardinalità**: un artista può (opzionalmente) pubblicare degli album o canzoni *(0,n)*, un album o canzone invece possono appartenere anche a più artisti *(1,m)*.
- **Composizione**: indica le *canzoni* presenti in un *album*.
  - **Cardinalità**: in un album ci possono essere una o più canzoni *(1,n)*, una canzone a sua volta invece può essere contenuta in un solo album *(1,1)*.
- **Pubblicazione podcast**: associa ad uno o più *podcaster* i *podcast* da essi pubblicati.
  - **Cardinalità**: un podcaster può (opzionalmente) pubblicare dei podcast *(0,n)*, un podcast invece può appartenere anche a più podcaster *(1,m)*.
- **Contenuto podcast**: indica gli *episodi* presenti in un *podcast*.
  - **Cardinalità**: in un podcast ci possono essere uno o più episodi *(1,n)*, un episodio a sua volta invece appartiene ad un solo podcast *(1,1)*.
- **Contenuto podcast**: indica gli *episodi* presenti in un *podcast*.
  - **Cardinalità**: in un podcast ci possono essere uno o più episodi *(1,n)*, un episodio a sua volta invece appartiene ad un solo podcast *(1,1)*.

- **Iscrizione**: descrive a quale *piano* si è iscritto l'*utente*.
  - **Cardinalità**: un utente può iscriversi ad un solo piano *(1,1)*, mentre per il piano ci sono vari casi:
    - **FREE**: possono essere associati ad un solo utente
    - **PREMIUM**: è associato a più utenti, fino ad un massimo di 4 *(1,n)*
- **Tipo**: indica a quale tipo di *abbonamento* si riferisce un *piano*.
  - **Cardinalità**: un piano ha un solo tipo di abbonamento *(1,1)*, mentre un tipo di abbonamento può essere presente in più piani *(1,n)*.
- **Modalità**: descrive il metodo di *pagamento* selezionato dall'*utente*.
  - **Cardinalità**: all'interno dei dati di fatturazione è associata una singola carta di credito *(1,1)*, mentre una carta di credito può essere presente in più profili utente *(1,n)*.
- **Estremi**: associa un *utente* ai *dati di fatturazione* che saranno riportati sulla fattura.
  - **Cardinalità**: ad un utente sono richiesti i dati di fatturazione se e solo se ha sottoscritto un abbonamento a pagamento (0,1), mentre i dati della fattura sono associati ad un singolo utente (1,1).
- **Fattura**: associa i *dati di fatturazione* alle informazioni riportate nel pagamento.
  - **Cardinalità**: i dati di fatturazione possono essere usati per effettuare più pagamenti (1,n), un pagamento viene effettuato con singoli dati di fatturazione (1,1).
- **Genere**: specifica le varie cateogorie a cui può appartenere un podcast.
  - **Cardinalità**: un podcast può essere di più categorie (1,n), mentre data una certa categoria, ci possono più podcast (0,m).

3 Progettazione concettuale

3.1 Lista Entità

- **Profilo**: entità generica necessaria per la creazione di un account, la generalizzazione è totale in quanto un profilo, una volta creato, necessariamente deve essere di una delle tipologie specifiche descritte successivamente.
  - **mail**: varchar (50) primary key
  - **nome**: varchar (50) not null
  - **cognome**: varchar (50) not null
  - **password**: varchar (30) not null
  - **followers**: int
- **Utente**: specializzazione dell'entità profilo, l'utente è un fruitore dei contenuti disponibili nella piattaforma.
  - **nickname**: varchar (20) not null unique
  - **following**: int
- **Artista**: specializzazione dell'entità profilo, creatore di contenuti musicali.
  - **nome_artista**: varchar (30) not null unique
  - **info**: varchar (200)
  - **label**: varchar (20)
  - **ascolti_mensili**: int

- **Podcaster**: specializzazione dell'entità profilo, creatore di podcast.
  - **nome podcaster**: varchar (30) not null unique
  - **info**: varchar (200)
- **Playlist**
  - **playlist_id**: char (10) primary key
  - **nome**: varchar (30) not null
  - **descrizione**: varchar (250)
  - **data_creazione**: date not null
  - **utente**: varchar (50) not null
- **Contenuto Playlist**
  - **playlist**: char (10) not null
  - **titolo**: varchar (30)
  - **album**: int
- **Canzone**
  - **titolo**: varchar (30) primary key
  - **album**: int (30) primary key (album non può avere valore nullo, in quanto non può esistere una singola canzone non appartenente ad esso)
  - **durata**: int not null
- **Album**
  - **album_id**: SERIAL primary key
  - **titolo**: varchar (30) not null
  - **artista**: varchar (50) not null (artista non può avere valore nullo, poiché un album necessariamente deve essere creato da un'artista)
  - **data_pubblicazione**: date not null
- **Piano**
  - **utente**: varchar(50) primary key
  - **inizio_piano**: date not null
  - **fine_piano**: date not null check

    ```
    (EXTRACT (MONTH FROM fine_piano) - EXTRACT (MONTH FROM inizio_piano) >= 1
    OR EXTRACT (YEAR FROM fine_piano) - EXTRACT (YEAR FROM inizio_piano) >= 1)
    ```

  - **abbonamento**: varchar (20)
- **Abbonamento**
  - **nome**: varchar(20) primary key check (nome in('FREE', 'PREMIUM', 'FAMILY'))
  - **costo_mensile**: numeric (4, 2) check

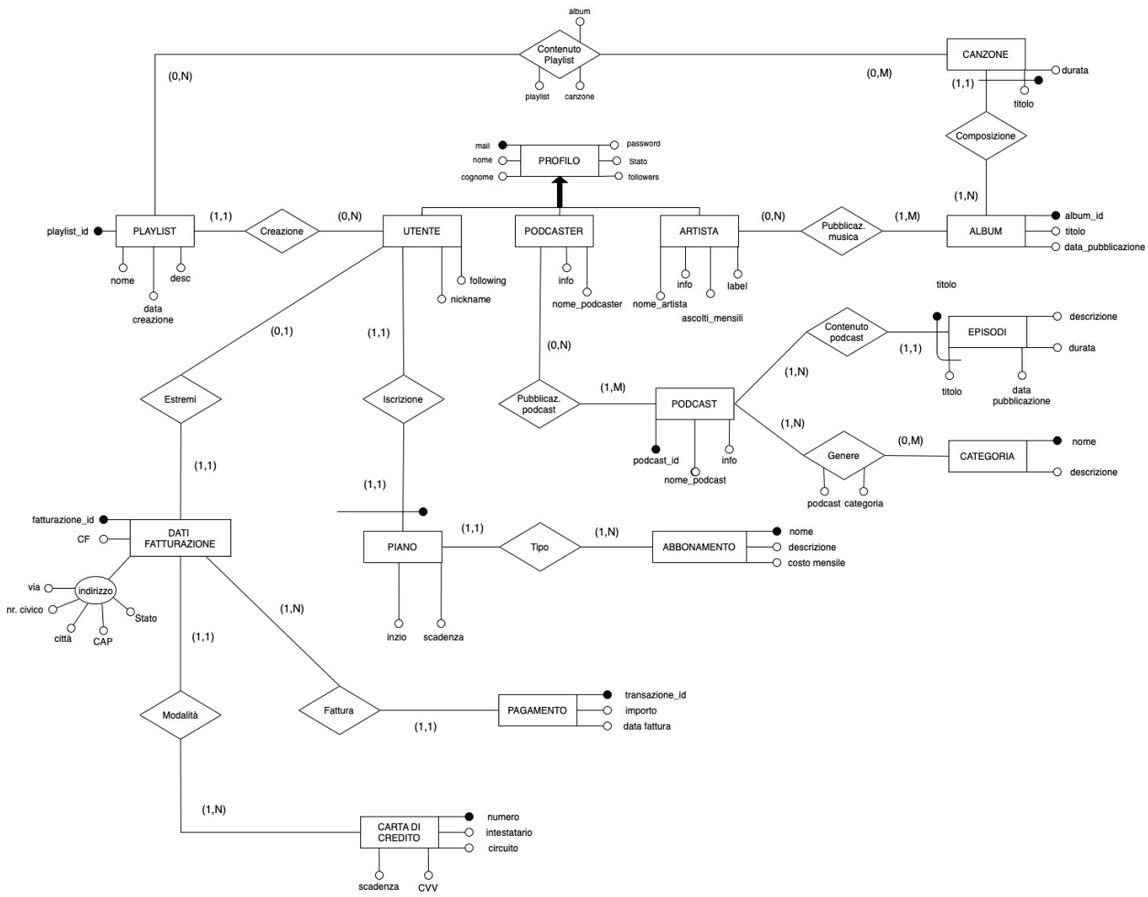
    ```
    (costo_mensile >= 0)
    ```

  - **descrizione**: varchar(200)

- **Dati Fatturazione**
  - **fatturazione id**: char(10) primary key
  - **codice_fiscale**: char(16) unique not null
  - **civico**: int not null
  - **stato**: char(2) not null
  - **cap**: int check (cap between 10000 and 99999) not null
  - **citta**: varchar(30) not null
  - **via**: varchar(30) not null
- **Carta Credito**
  - **numero**: numeric(16,0) primary key
  - **cvv**: numeric(3,0) not null
  - **intestatario**: varchar(50) not null
  - **scadenza**: date not null
  - **circuito**: varchar (20) not null check (circuito in('mastercard', 'visa', 'maestro'))
- **Pagamento**
  - **transazione id**: char (10) primary key
  - **fatturazione id**: char (10) not null
  - **data_fattura**: date not null
  - **importo**: numeric (4, 2) not null

    `check(importo >= 0)`
- **Podcast**
  - **podcast_id**: SERIAL primary key
  - **nome_podcast**: VARCHAR(30) not null
  - **info**: VARCHAR(100)
  - **podcaster**: VARCHAR(50) not null
- **Episodio**
  - **podcast**: int primary key
  - **titolo**: VARCHAR(50) not null primary key
  - **descrizione**: VARCHAR(300)
  - **durata**: int not null
  - **data_pubblicazione**: date not null
- **Genere**
  - **podcast**: int not null
  - **categoria**: varchar(30) not null
- **Categoria**
  - **nome**: varchar (30) primary key
  - **descrizione**: varchar(100)

3.2 Schema E-R



4 Progettazione logica

4.1 Analisi delle ridondanze

L'attributo importo, facente classe dell'entità pagamento, presenta una ridondanza in quanto il valore che rappresenta è facilmente ricavabile dalla moltiplicazione tra il costo mensile e la differenza in mesi tra i due attributi scadenza e inizio dell'entità piano.

Questa variazione comporta notevoli benefici: permette di risparmiare spazio occupato e rende il database più robusto ad eventuali errori.

Dall'altro lato però comporta un notevole aumento di accessi alle entità coinvolte per il calcolo di importo.

Supponendo di avere 10.000 di utenti, e che il 70% di essi decida di sottoscrivere un abbonamento mensile (indipendentemente dal fatto che sia gratuito o a pagamento), mentre il restante 30% selezioni una durata maggiore, si può quindi stimare che almeno 7.000 volte al mese l'attributo importo debba essere visionato/calcolato per permettere all'utente di pagare la cifra dovuta.

4.2 Operazioni

- L'accesso all'importo da pagare, implica ogni volta il calcolo dell'importo nel caso non sia presente la ridondanza

  - **Caso ridondanza assente**

Tavola dei volumi		
Concetto	Tipo	Volume
Pagamento	E	50.000
Utente	E	10.000
Dati Fatturazione	E	8.000
Piano	E	8.000
Abbonamento	E	3
Fattura	R	50.000
Estremi	R	7.000
Iscrizione	R	10.000
Tipo	R	8.000

Tavola delle operazioni		
Operazione	Tipo	Frequenza
Accesso all'importo da pagare	Interattiva	7.000 al mese

Tavola degli accessi			
Concetto	Costrutto	Accessi	Tipo
Pagamento	Entità	1	S
Utente	Entità	1	L
Dati Fatturazione	Entità	1	L
Piano	Entità	1	L
Abbonamento	Entità	1	L
Fattura	Relazione	1	L
Estremi	Relazione	1	L
Iscrizione	Relazione	1	L
Tipo	Relazione	1	L

    - **Costo operazione**: basandosi sulla tavola degli accessi e la frequenza dell'operazione 1, risulterà che si avranno 56.000 accessi in lettura e 7.000 in scrittura al mese.

- **Caso ridondanza presente**
    - L'attributo importo occupa 8 byte, moltiplicato per il numero di occorrenze di Pagamento equivale a circa 400 kilobyte di memoria aggiuntiva occupata.

Tavola dei volumi		
Concetto	Tipo	Volume
Utente	E	10.000
Dati Fatturazione	E	8.000
Pagamento	E	50.000
Estremi	R	7.000
Fattura	R	50.000

Tavola delle operazioni		
Operazione	Tipo	Frequenza
Accesso all'importo da pagare	Interattiva	7.000 al mese

Tavola degli accessi			
Concetto	Costrutto	Accessi	Tipo
Utente	Entità	1	L
Dati Fatturazione	Entità	1	L
Pagamento	Entità	1	L
Estremi	Relazione	1	L
Fattura	Relazione	1	L

    - **Costo operazione**: basandosi sulla tavola degli accessi e la frequenza dell'operazione 1, risulterà che si avranno 35.000 accessi in lettura.

- **Conclusione**: confrontando le due analisi si è concluso che la presenza di ridondanza riduce il numero degli accessi in maniera considerevole al costo irrisorio di memoria aggiuntiva occupata.

4.3 Eliminazione delle generalizzazione

In quanto lo schema concettuale presenta una generalizzazione totale, è conveniente effettuare un accorpamento del genitore con le entità figlie, dato che si otterrebbe un risparmio di memoria sostanzioso, non essendoci valori nulli, riducendo inoltre il numero di accessi.

Se invece adottassimo l'approccio opposto, ossia l'accorpamento delle entità figlie con il genitore, perderemmo il beneficio riguardante il risparmio di memoria sopracitato; mentre la sostituzione della generalizzazione con associazioni, conveniente nel caso in cui si abbia una generalizzazione parziale, è sconsigliata in quanto comporterebbe la visita al genitore per ogni accesso alle entità figlie, causando così un notevole aumento degli accessi.

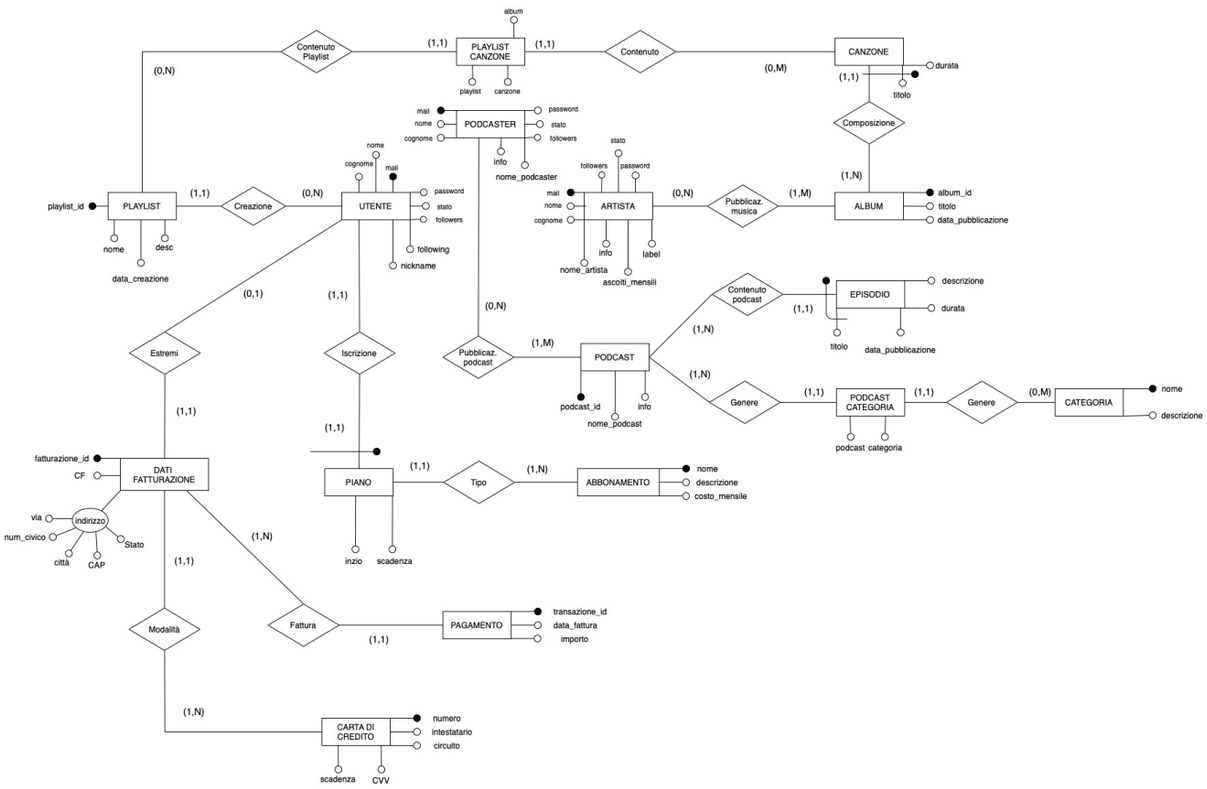
4.4 Scelta degli identificatori primari

Gli identificatori primari utilizzati nel database sono i seguenti:

- **album_id**: semplifica la referenziazione con l'entità canzone, in quanto può essere utilizzato come singola chiave esterna.

- **playlist_id**: rappresenta univocamente una singola playlist di un utente, alleggerendo così la referenziazione con le canzoni nel momento in cui esse vengono aggiunte in una playlist.
- **podcast_id**: concetto del tutto simile ad album_id, comporta gli stessi benefici.
- **fatturazione_id**: identifica, nel complesso, i dati di fatturazione generali inseriti dagli utenti. Semplifica la referenziazione con l'entità pagamento, ossia le singole transazioni effettuate per sottoscrivere un abbonamento sulla piattaforma.
- **piano_id**: rappresenta univocamente il piano di abbonamento selezionato dall'utente

4.5 Schema E-R ristrutturato



4.6 Descrizione schema relazionale e vincoli di integrità referenziale

- **Utente** (mail, nome, cognome, stato, password, nickname, followers, following)
- **Artista** (mail, nome, cognome, stato, password, followers, nome_artista, info, label, ascolti_mensili)
- **Podcaster** (mail, nome, cognome, stato, password, followers, nome podcaster, info)
- **Album** (album_id, titolo, artista, data_pubblicazione)
  Album.artista → Artista.mail
- **Podcast** (podcast_id, nome podcast, podcaster, info)
  Podcast.podcaster → Podcaster.mail
- **Canzone** (titolo, album, durata)
  Canzone.album → Album.album_id
- **Episodio** (podcast, titolo, descrizione, durata, data_pubblicazione)
  Episodio.podcast → Podcast.podcast_id
- **Abbonamento** (nome, costo_mensile, descrizione)
- **Playlist** (playlist_id, nome, descrizione, data_creazione, utente)
  Playlist.utente → Utente.mail
- **Contenuto_Playlist** (playlist, titolo, album)
  Contenuto_Playlist.playlist → Playlist.playlist_id
  Contenuto_Playlist.titolo → Canzone.titolo
  Contenuto_Playlist.album → Canzone.album

- **Piano** (utente, inizio_piano, fine_piano, abbonamento)
  Piano.utente → Utente.mail
  Piano.abbonamento → Abbonamento.nome

- **Categoria** (nome, descrizione)

- **Podcast_categoria** (podcast, categoria)
  Podcast_categoria.podcast → Podcast.podcast_id
  Podcast_categoria.categoria → Categoria.nome

- **Carta_di_Credito** (numero, cvv, intestatario, scadenza, circuito)

- **Dati_fatturazione** (fatturazione_id, codice_fiscale, civico, via, citta, cap, stato, utente, carta_credito)
  Dati_fatturazione.utente → Utente.mail
  Dati_fatturazione.carta_credito → Carta_di_Credito.numero

- **Pagamento** (transazione_id, fatturazione_id, data_fattura, importo)
  Pagamento.fatturazione_id → Dati_fatturazione.fatturazione_id

5 Queries

- **Query 1**: contare quante canzoni appartenenti ad uno stesso artista compaiono almeno due volte nelle playlist

```
SELECT artista.nome_artista, count(canzone.titolo)
FROM Contenuto_playlist, canzone, album, artista
WHERE Contenuto_playlist.album = canzone.album
AND contenuto_playlist.titolo = canzone.titolo
AND canzone.album = album.album_id
AND album.artista = artista.mail
GROUP BY artista.nome_artista
HAVING count(canzone.titolo) > 1
```

nome_artista	count
Rwidish	2
Posne	2
Fautied	3
Dogep	2
Cashbee	2
Bungeyd	5
Smallisonh	3
Rlaffin	5
Zbayldon	4
Jammet	2
Oquarmbyl	3
Murbyi	5
Odonett	4
Rhinckesmanc	2
Lwatsam	3

- **Query 2**: contare quanti album ha pubblicato un artista e mostrare il suo nome

```
SELECT artista.nome_artista, count(album.titolo) AS num_album
FROM album, artista
WHERE album.artista = artista.mail
GROUP BY artista.nome_artista
ORDER BY num_album ASC
```

nome_artista	num_album
Bmilborna	1
Ecrosdillw	1
Patriak	1
Kpieche	1
Ballintynec	1
Jammet	1
Slowderg	1
Acasseldine	2
Agaspard	2
Posne	2
Moralesr	2
Fagiolfingero	2
Nhaslinm	2
Smallisonh	2
Murbyi	2

- **Query 3**: mostrare il nome dell'album e dell'artista con il numero di canzoni in ordine decrescente

```
SELECT album.titolo, artista.nome_artista, count(canzone.titolo)
AS num_canzoni
FROM album, artista, canzone
WHERE album.artista = artista.mail
AND album.album_id = canzone.album
GROUP BY album.titolo, artista.nome_artista
ORDER BY num_canzoni DESC
```

titolo	nome_artista	num_canzoni
sapien	Grabiers	19
curabitur	Bungeyd	19
convallis duis	Moralesr	11
integer	Bungeyd	9
imperdiet	Odonett	9
orci mauris	Bungeyd	9
nulla quisque	Kpieche	8
sapien	Rwidish	8
libero ut massa	Grabiers	8
in leo	Rhinckesmanc	8
sit amet	Rlaffin	8
laoreet ut rhoncus	Bkirkmanb	8
ligula	Ubrum	7
in	Odonett	7
id turpis	Ehaslewooda	7
est	Clubbe	7

- **Query 4**: mostrare il nome del podcaster e del suo podcast con il numero di episodi in ordine decrescente

```
SELECT podcaster.nome, podcast.nome_podcast, count(episodio.titolo)
AS num_episodi
FROM podcaster, podcast, episodio
WHERE podcaster.mail = podcast.podcaster
AND podcast.podcast_id = episodio.podcast
GROUP BY podcaster.nome, podcast.nome_podcast
ORDER BY num_episodi DESC
```

nome	nome_podcast	num_episodi
Margot	tortor eu	11
Fowler	dui	11
Ann	mauris sit	10
Ann	enim	10
Bartholemy	nibh	10
Nichols	nascetur ridiculus	9
Cassandre	dolor vel	9
Sayre	turpis a	9
Elisabeth	quis	8
Ewen	suscipit a feugiat	8
Granger	lacinia	7
Margot	velit eu	7
Reggis	vitae ipsum	7
Matthieu	cras non velit	7
Marley	semper interdum	7
Marjie	vivamus vestibulum sagittis	7
Nichols	lobortis vel dapibus	7
Maure	sem	7
Urson	ultrices	7
Kean	dis	7
Granger	in consequat ut	7
Swen	at feugiat non	6

- **Query 5**: mostrare il nickname dell'utente e contare il numero di pagamenti effettuati

```
SELECT utente.nickname, count(pagamento.transazione_id) AS num_pagamenti
FROM utente, pagamento, dati_fatturazione
WHERE utente.mail = dati_fatturazione.Utente
AND dati_fatturazione.fatturazione_id = pagamento.fatturazione_id
GROUP BY utente.nickname
ORDER BY num_pagamenti DESC
```

nickname	num_pagamenti
jkermith	3
aabbie1	2
pspellacey3	1
ogriniove	1
lboughtflowerx	1
mokelleheru	1
ljowittn	1
darstallt	1
esyslands	1
tluetkemeyer5	1
afoxallf	1
lpinkettw	1
ahuycheg	1
ksecret8	1
dscholfield2	1
wmalyk	1
kmalyj4	1
escoblei	1
msier0	1
jtimlettj	1

6 Indici

In questo specifico caso di database, l'entità che potenzialmente può contenere un numero di record molto superiore rispetto alla media è "canzone". Si può immaginare il seguente scenario: l'utente ha intenzione di aggiungere una canzone in una sua playlist, dovendo quindi prima effettuare una ricerca che potrebbe essere inefficiente in termini di tempo e successivamente e l'inserimento nella tabella "contenuto_playlist", Per risolvere questo problema è possibile creare un indice riguardante gli attributi "titolo" e "album", in modo tale da migliorare le prestazioni dell'operazione complessiva.

```
CREATE INDEX canzone_indice ON canzone (titolo, album);
```

7 Codice C++

Per compilare correttamente il codice C++ incluso nel progetto, è sufficiente creare nello stesso path una cartella "dependecies", al cui interno contiene altre due sottocartelle "include" e "lib". Successivamento vanno rispettivamente inseriti i file: nella prima

```
libpq-fe.h

pg_config_ext.h

postgres_ext.h
```

mentre nella seconda

```
libpq.dll

libpq.lib
```

Una volta fatto ciò, eseguire il seguente comando:

```
g++ codice.cpp -L dependencies\lib -lpq -o codice
```

Per avviare il programma basta eseguire il file codice.exe appena generato, verrà richiesto l'inserimento di alcuni parametri, necessari per effettuare la connessione al database:

- il nome con il quale è stato creato il database
- username
- password (entrambi riferiti al proprietario del database)

Completato ciò, nel codice è presente una funzione che controllerà la validità dei parametri inseriti in input, ovvero che la connessione al database sia avvenuta con successo.
Verrà poi stampato un menù con la possibilità di scegliere, digitando un numero da 1 a 5, quale query eseguire e visualizzarne il risultato; in alternativa, per uscire dal programma, è sufficiente premere il numero 6.
Per ogni query è presente una funzione che ne controllerà lo stato del risultato prima di stampare a video i risultati.