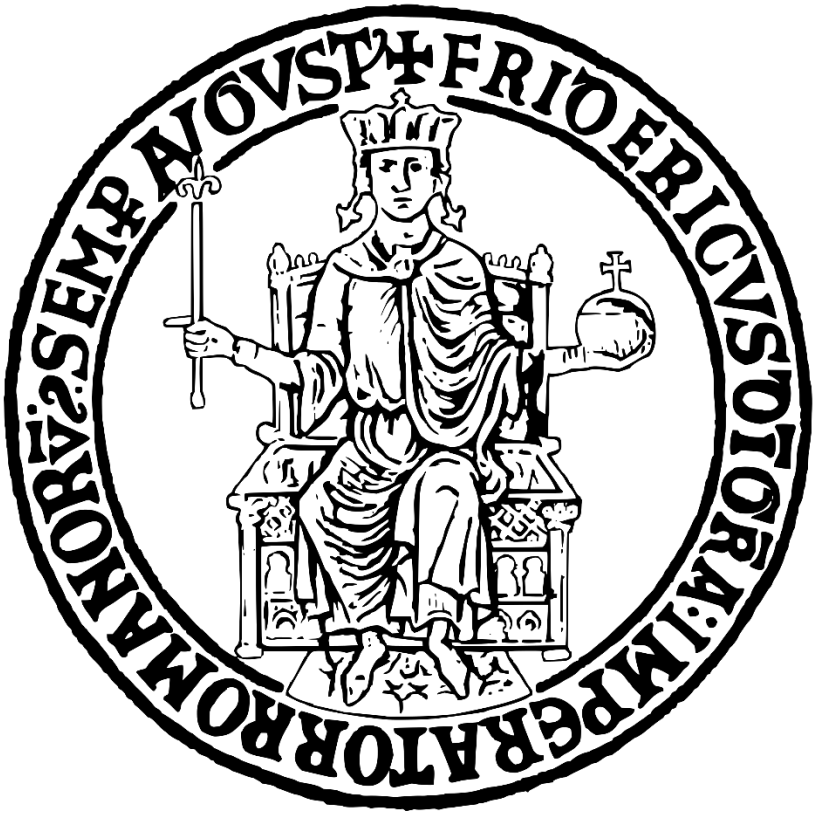
UNIVERSITÀDEGLI STUDI DI NAPOLI FEDERICO II

SCUOLA POLITECNICA E DELLE SCIENZE DI BASE

DIPARTIMENTO DI IGNEGNERIA ELETTRICA E TECNOLOGIE DELL'INFORMAZIONE



BISCIORSOUND

SISTEMA DI GESTIONE LIBRERIA MUSICALE DI UN GRUPPO DI UTENTI

TRACCIA 1

Vincenzo Meloni : N86003897

Loris Zannini : N86003744

Anno Accademico 2021/22

Docente:

Silvio Barra

INDICE

Capitolo1

Introduzione

1.1 Descrizione del progetto

BisciorSound è un sistema informativo, composto da una base di dati relazionale e da un applicativo Java dotato di GUI , per la gestione delle librerie musicali di un gruppo di utenti. Il sistema permette di gestire le singole tracce degli utenti, gli album di appartenenza ,gli ascolti delle tracce, i remastering , le cover , una lista di preferiti delle tracce e delle cover che l'utente decide di salvare. L'utente può accedere nella home, dove andrà ad interagire con la libreria musicale , effettuando il log-in o registrandosi (qualora non fosse dotato di account) inserendo il proprio username e la password. L'Admin è già registrato nel sistema e vi accederà tramite un'interfaccia a parte che lo reindirizzerà nella home Admin , permettendogli di effettuare le stesse operazioni di un normale utente con l'aggiunta di una sezione riservata esclusivamente ad egli, le "Opzioni Admin" , dentro le quali si potranno visionare tutti gli utenti registrati nel database (inclusi gli admin). Inoltre un amministratore avrà la possibilità di visionare , tramite una ricerca testuale , gli utenti che hanno effettuato più ascolti data una determinata traccia e potrà individuare la fascia oraria dove un preciso utente ha effettuato più ascolti differenziando tracce e cover (se nel sistema è presente la cover di un determinato brano).

1.2 Requisiti Identificati

Il primo requisito identificato dopo aver costruito la tabella "Utente" è stato quello di inserire al suo interno una variabile booleana che verifica se il medesimo è o non è un "Admin". Si è evitato di creare un'altra entità chiamata "Admin" , sostanzialmente per i seguenti motivi:

- Maggiore Chiarezza e Semplicità: Dato che nella tabella utente ci sono gli attributi associato ad esso, tra cui la chiave primaria "user_id" , per effettuare un controllo di "checkLogin" ad esempio, è più chiaro e semplice verificare se l'utente è o non è un admin andando ad effettuare una Query dove se è un utente comune la variabile booleana sarà false, altrimenti sarà true.
- Riduzione Delle Ridondanze: Nel database , così facendo , si è evitato di andare a creare un'ulteriore tabella per la manipolazioni dei dati di un utente che può svolgere particolari funzioni, andando ad inserire semplicemente l'attributo nell'entità Utente. Se avessimo creato un'ulteriore entità , avrebbe avuto gli stessi attributi di un normale utente, quindi per compattezza abbiamo preferito ridurre le ridondanze di attributi simili.

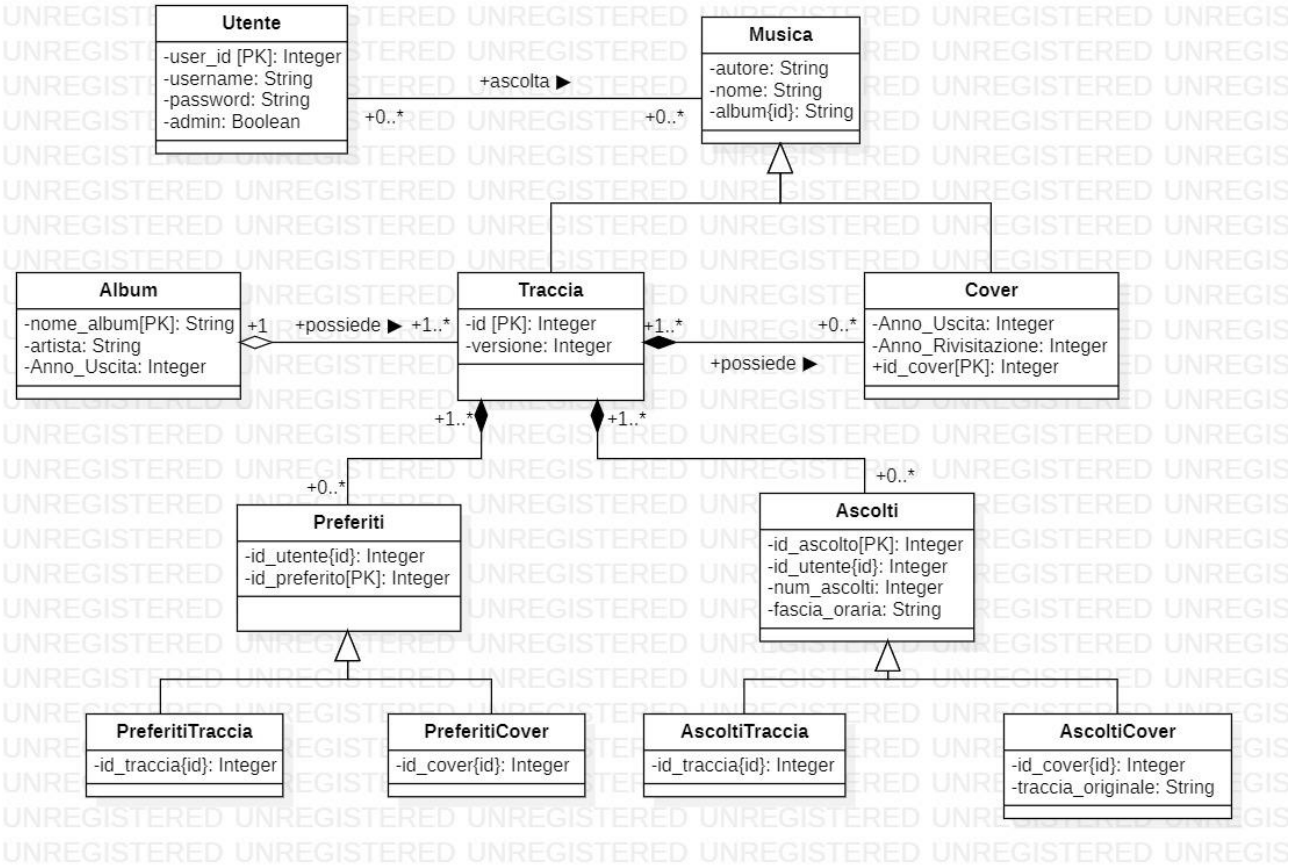
Per quanto riguarda i preferiti e gli ascolti, abbiamo creato due diverse tabelle per ognuno di loro: "Preferiti_traccia,Preferiti_cover" e "Ascolto_traccia, Ascolto_Cover" . Non è stato possibile inglobare tracce e cover in un'unica entità, poiché banalmente un utente può scegliere

di salvare nei preferiti una traccia ma non una cover e viceversa, inoltre può scegliere di ascoltare una traccia ma non una cover e viceversa. Ricordiamo che la cover non può esistere senza la traccia originaria. Questo implica che nelle tabelle ci siano diversi attributi per identificare una traccia o una cover. Da questo ragionamento deriva anche il fatto di aver differenziato le entità di Traccia e Cover per lo stesso motivo detto in precedenza. Così facendo, nella base di dati c'è maggiore chiarezza e di conseguenza maggiore semplicità nella stesura di Query sql. Un'ulteriore entità creata , correlata alle Tracce, è "Album", poiché ogni traccia ha un solo album di appartenenza, fatta eccezione per i remastering dato che remastering diversi della stessa traccia possono far partedi album diversi.

Capitolo 2

Progettazione Concettuale

2.1 Class Diagram



2.2 Ristrutturazione Del Class Diagram

Un passo importante è la ristrutturazione del Class Diagram , questo procedimento è necessario per migliorare l'efficienza dell'implementazione e per una corretta traduzione dello stesso in schemi relazionali. Si effettuano dei controlli sui parametri al fine di eliminare eventuali gerarchie di specializzazione, attributi strutturati e attributi multipli.

2.3 Analisi Delle Ridondanze

Non sono state riscontrate significative ridondanze da eliminare.

2.4 Analisi Degli Identificativi

Al fine di una corretta interpretazione e implementazione dell'applicativo, abbiamo utilizzato identificativi numerici che fungono da chiave primaria , come ad esempio id_utente ,che essendo univoco in tutto il sistema,permette di comprendere subito a quale utente specifico stiamo facendo riferimento. Medesima cosa per le altre entità, in modo tale da utilizzare con più facilitàle chiavi esterne ed avere maggiore chiarezza sulle istanze alle quali facciamo riferimento. L'unico identificativo non numerico è "nome_album" che in album funge da chiave primaria , e in traccia da chiave esterna per avere una corretta relazione tra una traccia e il rispettivo album di appartenenza.

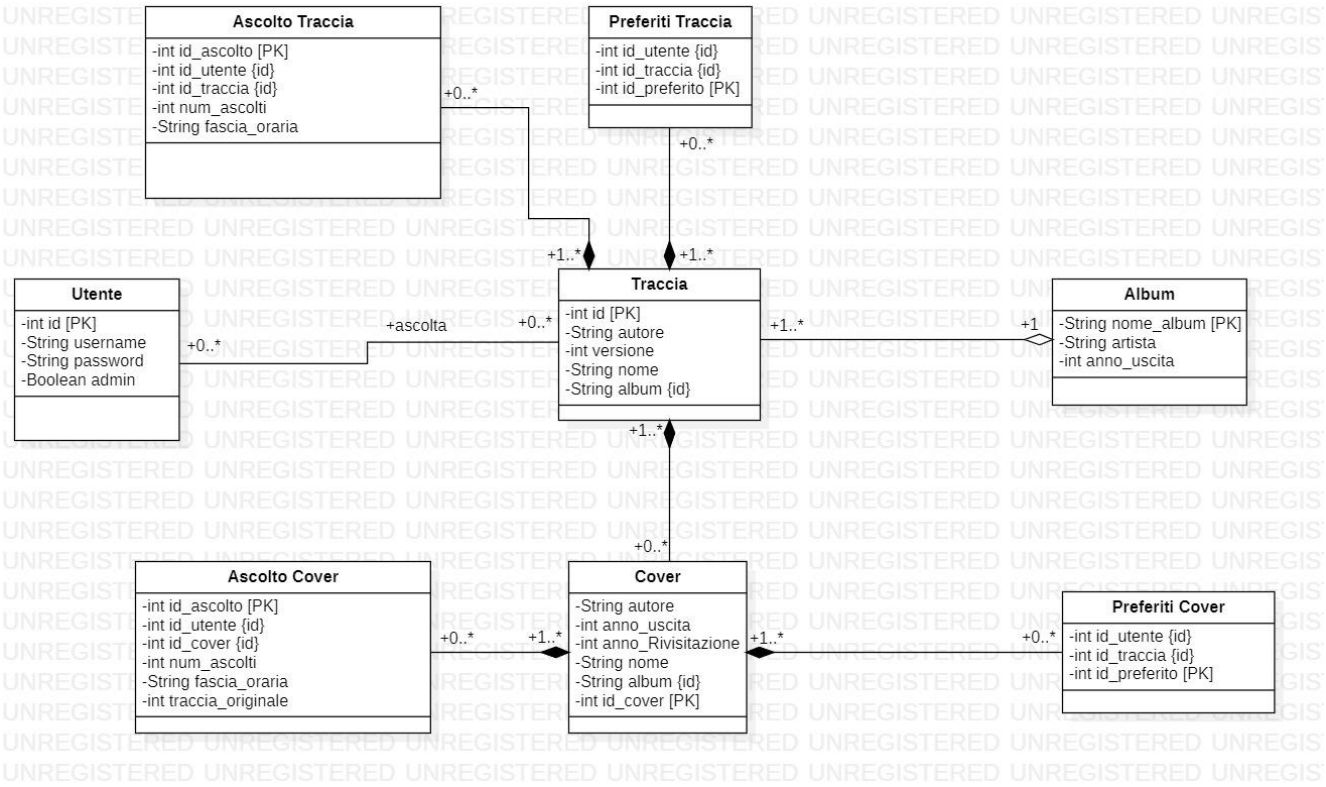
2.5 Rimozione Degli Attributi Multipli

Non sono presenti attributi multipli da eliminare.

2.6 Rimozione Delle Gerarchie Di Specializzazione

In primo luogo verrà effettuata la rimozione della classe "Musica" che è una generalizzazione di "Traccia" e "Cover". Eliminando quest'ultima andremo a manipolare i dati considerando le due sottoclassi come entità disgiunte, ricordando che una cover non può esistere senza la traccia originaria. Eliminando quest'ultima, andranno eliminate anche le successive gerarchie, cioè "preferiti" e "ascolti", componendo i preferiti delle tracce e delle cover con la classe di appartenenza ,medesima cosa per gli ascolti.

2.7 Class Diagram Ristrutturato



2.8 Dizionario Delle Classi

Classe	*Descrizione*	*Attributi*
Utente	Descrittore dell' Utente.	**user_id**(*Integer*): Chiave Primaria. Identifica univocamente un utente. **username**(*String*): nome associato all'utente. **password**(String): password associata all'username. **admin**(Boolean): verifica se l'utente è un admin
Traccia	Descrittore della Traccia.	**id_track**(*Integer*): Chiave Primaria. Identifica univocamente una traccia. **autore**(*String*): nome dell'autore associato alla traccia. **versione**(*Integer*): versione della traccia di riferimento. **nome**(*String*): nome della traccia. **album**(*String*): Chiave Esterna. Nome dell'album a cui fa riferimento la traccia.

Classe	*Descrizione*	*Attributi*
Album	Descrittore degli Album.	**nome_album**(*String*): Chiave Primaria. Nome associato all'album. **artista**(*String*): nome associato all'artista dell'album di riferimento. **anno_uscita**(*Integer*): anno di uscita (o nascita) dell'album di riferimento.
Cover	Descrittore della Cover	**autore**(*String*): nome associato all'artista. **anno_nascita**(*Integer*): anno associato all'uscita della traccia originaria. **anno_rivisitazione**(Integer): anno associato alla rivisitazione della traccia. **nome**(*String*): nome associato alla cover. **album**(*String*): Chiave esterna. nome associato all'album di riferimento. **id_cover**(*Integer*): Chiave Primaria. Identificativo associato alla cover ,univoco. **traccia_originale**(*Integer*): Chiave esterna. identificativo associato alla traccia originale.

Classe	*Descrizione*	*Attributi*
Preferiti Traccia	Descrittore delle tracce salvate nei Preferiti.	**id_utente***(Integer):*Chiave esterna. Identificativo associato all'utente. **id_traccia**(*Integer*): Chiave esterna. Identificativo associato alla traccia. **id_preferito**(*Integer*): Chiave Primaria. Identificativo associato ai preferiti.
Preferiti Cover	Descrittore delle Cover salvate nei Preferiti.	**id_utente***(Integer):*Chiave esterna. Identificativo associato all'utente. **id_cover**(*Integer*): Chiave esterna. Identificativo associato alla cover. **id_preferito**(*Integer*): Chiave Primaria. Identificativo associato ai preferiti.
Ascolto Traccia	Descrittore degli ascolti sulle tracce che sono state riprodotte.	**id_ascolto***(Integer):* Chiave Primaria. Identificativo associato all'ascolto di una traccia. **id_utente**(*Integer*): Chiave esterna. Identificativo associato all'utente di riferimento. **id_traccia**(*Integer*): Chiave esterna. Identificativo associato alla traccia di riferimento. **num_ascolti**(*Integer*): valore numerico associato a quante volte l'utente ha riprodotto una determinata traccia. **fascia_oraria**(*String*): fascia oraria in cui l'utente ha riprodotto una determinata traccia.

Classe	*Descrizione*	*Attributi*
Ascolto Cover	Descrittore degli ascolti delle Cover che sono state riprodotte.	**id_ascolto***(Integer):* Chiave Primaria. Identificativo associato all'ascolto di una cover. **id_utente**(*Integer*): Chiave esterna. Identificativo associato all'utente di riferimento. **id_cover**(*Integer*): Chiave esterna. Identificativo associato alla cover di riferimento. **num_ascolti**(*Integer*): valore numerico associato a quante volte l'utente ha riprodotto una determinata cover. **fascia_oraria**(*String*): fascia oraria in cui l'utente ha riprodotto una determinata cover.

2.9 Dizionario Delle Associazioni

Per motivi di maggiore leggibilità le associazioni qui elencate non sono state scritte all'interno del Class Diagram.

Associazione	*Descrizione*	*Classi coinvolte*
Ascolta	Esprime quali utenti hanno ascoltato una traccia.	**Traccia [0…*]** indica quale traccia ha ascoltato un utente. **Utente [0…*]** indica da quale utente è stata ascoltata una traccia.
Possiede	Esprime quali tracce possiede un album.	**Traccia [1…*]** indica le tracce che ha un album. **Album [1]** indica a quale album appartiene una traccia.
Ha_Cover	Esprime quali tracce hanno una cover.	**Traccia [1…*]** indica le tracce che hanno una cover. **Cover [0…*]** indica a quale cover fa riferimento una traccia.
Ha_AscoltiT	Esprime il numero di ascolti per una traccia.	**Traccia [1…*]** indica gli ascolti di una traccia per un insieme di utenti. **Ascolto_Traccia [0…*]** indica quali tracce sono state ascoltate.
Ha_PreferitiT	Esprime quali utenti hanno aggiunto ai preferiti una traccia.	**Traccia [1…*]** indica la lista preferiti di una traccia per un insieme di utenti. **Preferiti_Traccia[0…*]** indica quali tracce sono state aggiunte ai preferiti.
Ha_AscoltiC	Esprime il numero di ascolti per una cover.	**Cover [1…*]** indica gli ascolti di una cover per un insieme di utenti. **Ascolto_Cover [0…*]** indica quali cover sono state ascoltate.
Ha_PreferitiC	Esprime quali utenti hanno aggiunto ai preferiti una cover.	**Cover [1…*]** indica la lista preferiti di una cover per un insieme di utenti. **Preferiti_Cover [0…*]** indica quali cover sono state aggiunte ai preferiti.

2.10 Dizionario Dei Vincoli

Nome	*Descrizione*
Unique Utente	In Utente lo username è un attributo con constraint unique, poiché nel sistema non ci possono essere utenti che hanno lo stesso nome.
Unique Cover	In Cover l'attributo anno_nascita ha un constraint unique, poiché l'anno della traccia originale da cui nasce la cover deve essere unico.
controllo_anno_albumtraccia	Se una traccia ha un anno di uscita superiore rispetto a quello di uscita dell'album a cui appartiene, allora non può essere aggiunta al sistema.
controllo_autorecover	Se una cover ha lo stesso artista della traccia originale da cui è nata, allora non può essere aggiunta al sistema.
controllo_lunghezzapassword	In fase di registrazione, la password inserita dall'utente deve contenere almeno 6 caratteri.
controllo_preferiti_traccia	Se una traccia è già presente nella lista dei preferiti, allora non può essere aggiunta di nuovo.
controllo_preferiti_cover	Se una cover è già presente nella lista dei preferiti, allora non può essere aggiunta di nuovo.
controllo_sicurezzapassword	In fase di registrazione la password inserita dall'utente deve essere diversa da "123456", "12345678" o "000000".
controllo_username	In fase di registrazione, il nome utente inserito deve contenere almeno 6 caratteri.
Key Utente	L'attributo user_id in Utente è chiave primaria.
Key Traccia	L'attributo id_track in Traccia è chiave primaria, mentre album è chiave esterna e si riferisce all'attributo nome_album in Album.
Key Cover	L'attributo id_cover in Cover è chiave primaria; abbiamo due chiavi esterne: traccia_originale che si riferisce a id_track di Traccia e album che si riferisce a nome_album in Album.
Key Album	L'attributo nome_album in Album è chiave primaria.

Nome	*Descrizione*
Key preferiti_traccia	Abbiamo due chiavi esterne: id_traccia che si riferisce all'attributo id_track di Traccia e id_utente che si riferisce a user_id in Utente.
Key preferiti_cover	Abbiamo due chiavi esterne: id_cover che si riferisce all'attributo id_cover di Cover e id_utente che si riferisce a user_id in Utente.
Key ascolto_traccia	L'attributo id_ascolto è chiave primaria. Abbiamo due chiavi esterne: id_traccia che si riferisce a id_track di Traccia e id_utente che si riferisce a user_id in Utente.
Key ascolto_cover	L'attributo id_ascoltoc è chiave primaria. Abbiamo due chiavi esterne: id_cover che si riferisce a id_cover di Cover e id_utente che si riferisce a user_id in Utente.

Capitolo3

Progettazione Logica

In questo capitolo procediamo alla progettazione logica. Le chiavi primarie verranno evidenziate in **grassetto** , mentre le chiavi esterne verranno sottolineate.

3.1 Schema Logico

Utente (**user_id**, username, password, admin)

Traccia (**id_track**, autore, versione, album)

Cover(autore, anno_nascita, anno_rivisitazione, nome, album, **id_cover**,traccia_originale)

Album (**nome_album**, artista, anno_uscita)

Preferiti Traccia (id_utente, id_traccia, **id_preferito**)

Preferiti Cover (id_utente, id_cover, **id_preferito**)

Ascolto Traccia (**id_ascolto**, id_utente, id_traccia, num_ascolti, fascia_oraria)

Ascolto Cover (**id_ascolto**, id_utente, id_cover, num_ascolti, fascia_oraria)

Capitolo4

Progettazione Fisica

4.1 Introduzione

In questa fase si procede alla definizione delle tabelle, alle implementazioni dei vincoli e dei trigger. Riporteremo gli script estratti dall'inizio della creazione del database. In questo caso il DBMS utilizzato è PostgresSQL 14.

4.2 Definizione Tabelle

In questa fase, come accennato in precedenza, seguiranno le definizioni delle tabelle estratte dallo script di creazione del database.Le tabelle indicate nella progettazione logica che hanno uno spazio , ad esempio "Preferiti Traccia", per la corretta sintassi, sono state sostituite con un "underscore" (_). Quindi "Preferiti Traccia" diventerà "preferiti_traccia" , questo per tutte le tabelle che lo necessitano.

4.2.1 Definizione Tabella UTENTE

```
-- Table: public.utente


-- DROP TABLE IF EXISTS public.utente;


CREATE TABLE IF NOT EXISTS public.utente
(
user_id integer NOT NULL GENERATED ALWAYS AS IDENTITY ( INCREMENT 1 START 20
MINVALUE 20 MAXVALUE 100000000 CACHE 1 ),
username character varying(32) COLLATE pg_catalog."default" NOT NULL,
password character varying(20) COLLATE pg_catalog."default" NOT NULL,
    admin boolean NOT NULL DEFAULT false,
```

```
    CONSTRAINT "AUTO_INCREMENT" PRIMARY KEY (user_id),
    CONSTRAINT conun UNIQUE (username),
    CONSTRAINT utente_username_key UNIQUE (username)
)

-- Trigger: controllo_lunghezzapassword

-- DROP TRIGGER IF EXISTS controllo_lunghezzapassword ON public.utente;

CREATE TRIGGER controllo_lunghezzapassword
    AFTER INSERT
    ON public.utente
    FOR EACH ROW
EXECUTE FUNCTION public.controllo_lunghezzapassword();

-- Trigger: controllo_sicurezzapassword

-- DROP TRIGGER IF EXISTS controllo_sicurezzapassword ON public.utente;

CREATE TRIGGER controllo_sicurezzapassword
    AFTER INSERT
    ON public.utente
    FOR EACH ROW
    EXECUTE FUNCTION public.controllo_sicurezzapassword();

-- Trigger: controllo_username

-- DROP TRIGGER IF EXISTS controllo_username ON public.utente;

CREATE TRIGGER controllo_username
    AFTER INSERT
    ON public.utente
```

```
    FOR EACH ROW
    EXECUTE FUNCTION public.controllo_username();
```

4.2.2 Definizione Tabella TRACCIA

```
-- Table: public.traccia

-- DROP TABLE IF EXISTS public.traccia;

CREATE TABLE IF NOT EXISTS public.traccia
(
id_track integer NOT NULL,
autore character varying(32) COLLATE pg_catalog."default" NOT NULL,
versione integer NOT NULL,
nome character varying(32) COLLATE pg_catalog."default" NOT NULL,
album character varying(32) COLLATE pg_catalog."default" NOT NULL,
    CONSTRAINT traccia_pkey PRIMARY KEY (id_track),
    CONSTRAINT traccia_album_fkey FOREIGN KEY (album)
        REFERENCES public.album (nome_album) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
)

-- Trigger: controllo_anno_albumtraccia

-- DROP TRIGGER IF EXISTS controllo_anno_albumtraccia ON public.traccia;

CREATE TRIGGER controllo_anno_albumtraccia
AFTER INSERT
ON public.traccia
    FOR EACH ROW
EXECUTE FUNCTION public.controllo_anno_albumtraccia();
```

4.2.3 Definizione Tabella ALBUM

```
-- Table: public.album

-- DROP TABLE IF EXISTS public.album;

CREATE TABLE IF NOT EXISTS public.album
(
nome_album character varying(32) COLLATE pg_catalog."default" NOT NULL,
artista character varying(32) COLLATE pg_catalog."default",
anno_uscita integer NOT NULL,
CONSTRAINT album_pkey PRIMARY KEY (nome_album)
)
```

4.2.4 Definizione Tabella COVER

```
-- Table: public.cover
-- DROP TABLE IF EXISTS public.cover;
CREATE TABLE IF NOT EXISTS public.cover
(
autore character varying(32) COLLATE pg_catalog."default" NOT NULL,
anno_nascita integer NOT NULL,
    anno_rivisitazione integer NOT NULL,
nome character varying(32) COLLATE pg_catalog."default" NOT NULL,
album character varying(32) COLLATE pg_catalog."default",
id_cover integer NOT NULL,
traccia_originale integer NOT NULL,
CONSTRAINT cover_pkey PRIMARY KEY (id_cover),
CONSTRAINT uni UNIQUE (anno_nascita),
CONSTRAINT fk1 FOREIGN KEY (traccia_originale)
        REFERENCES public.traccia (id_track) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION,
    CONSTRAINT fk2 FOREIGN KEY (album)
        REFERENCES public.album (nome_album) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
)

-- Trigger: controllo_autorecover
-- DROP TRIGGER IF EXISTS controllo_autorecover ON public.cover;

CREATE TRIGGER controllo_autorecover
    AFTER INSERT
    ON public.cover
    FOR EACH ROW
    EXECUTE FUNCTION public.controllo_autorecover();
```

4.2.5 Definizione Tabella PREFERITI_TRACCIA

```
-- Table: public.preferiti_traccia


-- DROP TABLE IF EXISTS public.preferiti_traccia;


CREATE TABLE IF NOT EXISTS public.preferiti_traccia

(

id_utente integer NOT NULL,

id_traccia integer NOT NULL,

id_preferito integer NOT NULL GENERATED ALWAYS AS IDENTITY ( INCREMENT 1 START
54 MINVALUE 54 MAXVALUE 100000 CACHE 1 ),

    CONSTRAINT preferiti_id_traccia_fkey FOREIGN KEY (id_traccia)

        REFERENCES public.traccia (id_track) MATCH SIMPLE

        ON UPDATE NO ACTION

        ON DELETE NO ACTION,

    CONSTRAINT preferiti_id_utente_fkey FOREIGN KEY (id_utente)

        REFERENCES public.utente (user_id) MATCH SIMPLE

        ON UPDATE NO ACTION

        ON DELETE NO ACTION

)


-- Trigger: controllo_preferiti_traccia


-- DROP TRIGGER IF EXISTS controllo_preferiti_traccia ON
public.preferiti_traccia;


CREATE TRIGGER controllo_preferiti_traccia

AFTER INSERT

ON public.preferiti_traccia

    FOR EACH ROW

EXECUTE FUNCTION public.controllo_preferiti_traccia();
```

4.2.6 Definizione Tabella PREFERITI_COVER

```
-- Table: public.preferiti_cover

-- DROP TABLE IF EXISTS public.preferiti_cover;

CREATE TABLE IF NOT EXISTS public.preferiti_cover
(
id_utente integer NOT NULL,
id_cover integer NOT NULL,
id_preferito integer NOT NULL GENERATED ALWAYS AS IDENTITY ( INCREMENT 1 START
26 MINVALUE 26 MAXVALUE 100000 CACHE 1 ),
    CONSTRAINT preferiti_cover_pkey PRIMARY KEY (id_preferito),
    CONSTRAINT preferiti_cover_id_cover_fkey FOREIGN KEY (id_cover)
        REFERENCES public.cover (id_cover) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION,
    CONSTRAINT preferiti_cover_id_utente_fkey FOREIGN KEY (id_utente)
        REFERENCES public.utente (user_id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
)

TABLESPACE pg_default;

-- DROP TRIGGER IF EXISTS controllo_preferiti_cover ON public.preferiti_cover;

CREATE TRIGGER controllo_preferiti_cover
    AFTER INSERT
    ON public.preferiti_cover
    FOR EACH ROW
EXECUTE FUNCTION public.controllo_preferiti_cover();
```

4.2.7 Definizione Tabella ASCOLTO_TRACCIA

```
-- Table: public.ascolto_traccia


-- DROP TABLE IF EXISTS public.ascolto_traccia;


CREATE TABLE IF NOT EXISTS public.ascolto_traccia
(
id_ascolto integer NOT NULL GENERATED BY DEFAULT AS IDENTITY ( INCREMENT 1
START 75 MINVALUE 75 MAXVALUE 100000 CACHE 1 ),
id_utente integer NOT NULL,
id_traccia integer NOT NULL,
num_ascolti integer DEFAULT 0,
    fascia_oraria character varying(16) COLLATE pg_catalog."default",
CONSTRAINT ascolto_pkey PRIMARY KEY (id_ascolto),
    CONSTRAINT fk1 FOREIGN KEY (id_utente)
        REFERENCES public.utente (user_id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION,
    CONSTRAINT fk2 FOREIGN KEY (id_traccia)
        REFERENCES public.traccia (id_track) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
)
```

4.2.8 Definizione Tabella ASCOLTO_COVER

```
-- Table: public.ascolto_cover


-- DROP TABLE IF EXISTS public.ascolto_cover;


CREATE TABLE IF NOT EXISTS public.ascolto_cover
(
id_ascoltoc integer NOT NULL GENERATED BY DEFAULT AS IDENTITY ( INCREMENT 1
START 25 MINVALUE 25 MAXVALUE 100000 CACHE 1 ),
id_utente integer NOT NULL,
id_cover integer NOT NULL,
num_ascolti integer DEFAULT 0,
    fascia_oraria character varying(16) COLLATE pg_catalog."default",
CONSTRAINT ascolto_cover_pkey PRIMARY KEY (id_ascoltoc),
    CONSTRAINT ascolto_cover_id_cover_fkey FOREIGN KEY (id_cover)
        REFERENCES public.cover (id_cover) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION,
    CONSTRAINT ascolto_cover_id_utente_fkey FOREIGN KEY (id_utente)
        REFERENCES public.utente (user_id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
)
```

4.3 Funzioni, Procedure ed altre Automazioni

4.3.1 Controllo automatico lunghezza password

Il trigger **controllo_lunghezzapassword() ,** nel momento in cui un utente effettua la registrazione , viene attivato il trigger che controlla se la lunghezza della password è idonea, in questo caso deve essere minimo di 6 caratteri, in caso negativo viene sollevata un'eccezione. Di seguito è riportata la definizione:

```
-- FUNCTION: public.controllo_lunghezzapassword()


-- DROP FUNCTION IF EXISTS public.controllo_lunghezzapassword();


CREATE OR REPLACE FUNCTION public.controllo_lunghezzapassword()
    RETURNS trigger
    LANGUAGE 'plpgsql'
    COST 100
    VOLATILE NOT LEAKPROOF
AS $BODY$
DECLARE
pass_wordutente.password%type;
BEGIN
          SELECT utente.password INTO pass_word
          FROM utente
          WHERE utente.password = NEW.password;


          IF(LENGTH(pass_word) < 6) THEN
          RAISE EXCEPTION 'La password devecontenerealmeno 6 caratteri!'
            USING HINT = 'Inserisci una password più lunga.';
          END IF;
RETURN NULL;
END;


$BODY$
```

4.3.2 Controllo automatico Sicurezza Password

Il trigger **controllo_sicurezzapassword()**, viene attivato nel momento in cui un utente, in fase di registrazione, utilizza una password che viene reputata poco sicura. In questo caso viene sollevata un'eccezione che invita l'utente ad utilizzare una password più sicura. Di seguito è riportata la definizione:

```
-- FUNCTION: public.controllo_sicurezzapassword()


-- DROP FUNCTION IF EXISTS public.controllo_sicurezzapassword();


CREATE OR REPLACE FUNCTION public.controllo_sicurezzapassword()
    RETURNS trigger
    LANGUAGE 'plpgsql'
    COST 100
    VOLATILE NOT LEAKPROOF
AS $BODY$
DECLARE
pass_wordutente.password%type;
BEGIN
          SELECT utente.password INTO pass_word
          FROM utente
          WHERE utente.password = NEW.password;


          IF(pass_word = '123456' OR pass_word = '12345678' OR pass_word =
'000000' OR pass_word = 'password') THEN
          RAISE EXCEPTION 'La password inserita ha un livello di sicurezza
troppo basso!'
            USING HINT = 'Inserisci una password più sicura.';
          END IF;
RETURN NULL;
END;


$BODY$
```

4.3.3 Controllo automatico Preferiti Cover

Il trigger **controllo_preferiti_cover()**, viene attivato nel momento in cui un utente decide di aggiungere ai preferiti una cover che ha già precedentemente salvato, nel caso in cui la stessa è già presente, viene sollevata un'eccezione che avvisa l'utente. Di seguito è riportata la definizione:

```
-- FUNCTION: public.controllo_preferiti_cover()


-- DROP FUNCTION IF EXISTS public.controllo_preferiti_cover();


CREATE OR REPLACE FUNCTION public.controllo_preferiti_cover()
    RETURNS trigger
    LANGUAGE 'plpgsql'
    COST 100
    VOLATILE NOT LEAKPROOF
AS $BODY$
DECLARE

BEGIN

IF EXISTS(
    SELECT *
    FROM preferiti_cover AS p1, preferiti_cover AS p2
    WHERE p1.id_utente = p2.id_utente AND p1.id_cover = p2.id_cover AND
p1.id_preferito <> p2.id_preferito
)THEN
RAISE EXCEPTION 'La cover è già presente nei preferiti!';
END IF;

RETURN NULL;
END;

$BODY$;
```

4.3.4 Controllo automatico Preferiti Traccia

Il trigger **controllo_preferiti_traccia()**, viene attivato nel momento in cui un utente decide di aggiungere ai preferiti una traccia che ha già precedentemente salvato, nel caso in cui la stessa è già presente, viene sollevata un'eccezione che avvisa l'utente. Di seguito è riportata la definizione:

```
-- FUNCTION: public.controllo_preferiti_traccia()


-- DROP FUNCTION IF EXISTS public.controllo_preferiti_traccia();


CREATE OR REPLACE FUNCTION public.controllo_preferiti_traccia()
    RETURNS trigger
    LANGUAGE 'plpgsql'
    COST 100
    VOLATILE NOT LEAKPROOF
AS $BODY$
DECLARE

BEGIN

IF EXISTS(
    SELECT *
    FROM preferiti_traccia AS p1, preferiti_traccia AS p2
    WHERE p1.id_utente = p2.id_utente AND p1.id_traccia = p2.id_traccia and
p1.id_preferito <> p2.id_preferito
)THEN
RAISE EXCEPTION 'La traccia è già presente nei preferiti!';
END IF;

RETURN NULL;
END;

$BODY$;
```

4.3.5 Controllo automatico Username

Il trigger **controllo_username()**, si attiva in fase di registrazione, quando un utente , registrandosi, inserisce il proprio username, quest'ultimo deve contenere almeno 4 caratteri, in caso contrario viene sollevata un'eccezione. Di seguito è riportata la definizione:

```
-- FUNCTION: public.controllo_username()


-- DROP FUNCTION IF EXISTS public.controllo_username();


CREATE OR REPLACE FUNCTION public.controllo_username()
    RETURNS trigger
    LANGUAGE 'plpgsql'
    COST 100
    VOLATILE NOT LEAKPROOF
AS $BODY$
DECLARE
user_nameutente.username%type;
BEGIN
          SELECT utente.username INTO user_name
          FROM utente
          WHERE utente.username = NEW.username;


          IF(LENGTH(user_name) < 4) THEN
          RAISE EXCEPTION 'Il nome utente deve contenere almeno 4 caratteri!'
            USING HINT = 'Inserisci un nome utente più lungo.';
          END IF;
RETURN NULL;
END;


$BODY$;
```

4.3.6 Controllo automatico Anno Traccia e Album

Il trigger **controllo_anno_albumtraccia()**, si attiva quando si aggiunge una traccia al sistema. Il suo scopo è quello di verificare che l'anno della traccia inserita sia minore o uguale al suo album di appartenenza, poiché una traccia non può nascere dopo la pubblicazione dell'album che la contiene. Di seguito è riportata la definizione:

```
-- FUNCTION: public.controllo_anno_albumtraccia()

-- DROP FUNCTION IF EXISTS public.controllo_anno_albumtraccia();


CREATE OR REPLACE FUNCTION public.controllo_anno_albumtraccia()

    RETURNS trigger

    LANGUAGE 'plpgsql'

    COST 100

    VOLATILE NOT LEAKPROOF

AS $BODY$

DECLARE

annoAalbum.anno_uscita%TYPE;

annoTtraccia.versione%TYPE;

BEGIN

    SELECT Traccia.versione  INTOannoT

    FROM Traccia

WHERE Traccia.nome = NEW.nome;


SELECT album.annouscita INTO annoA

    FROM album

WHERE album.nome = NEW.album;


IF(annoT>annoA) THEN

          RAISE EXCEPTION 'L''anno della traccia non può essere maggiore
all''anno di uscita dell''album di appartenenza!';

          END IF;

RETURN NULL;

END;

$BODY$;
```

4.3.7 Controllo automatico Autore Cover

Il trigger **controllo_autorecover()**, si attiva dopo l'inserimento di una cover nel sistema. Il suo scopo è quello di verificare che l'autore della cover sia diverso dall'autore della traccia originale, poiché una cover è la riesecuzione di una traccia già esistente, ma da parte di un altro artista. Di seguito è riportata la definizione:

```
-- FUNCTION: public.controllo_autorecover()

-- DROP FUNCTION IF EXISTS public.controllo_autorecover();


CREATE OR REPLACE FUNCTION public.controllo_autorecover()

    RETURNS trigger

    LANGUAGE 'plpgsql'

    COST 100

    VOLATILE NOT LEAKPROOF

AS $BODY$

DECLARE

autoreCcover.autore%TYPE;

autoreTtraccia.autore%TYPE;

BEGIN

    SELECT cover.autore  INTOautoreC

    FROM Cover

    WHERE Cover.nome = NEW.nome;


    SELECT traccia.autore INTO autoreT

    FROM Traccia

WHERE Traccia.nome LIKE '%NEW.nome%';


IF(autoreC = autoreT) THEN

          RAISE EXCEPTION 'L''autore della cover deve essere diverso
dall''autore della traccia!';

          END IF;

RETURN NULL;

END;

$BODY$;
```

Capitolo5

Manuale d’Uso

5.1 Popolamento con Dati di Esempio

Di seguito è riportato lo script di popolamento della base di dati, con alcuni dati di esempio.

UTENTE:

```
INSERT INTO UTENTE("user_id","username","password","admin")
VALUES(1,"lollo01","informatica",True),
(2,"fNapoli26","SSCNapoli",False),
(3,"mario_","mm2000",False),
(4,"xskript","Musica1",False),
(5,"VincyM","database",True),
(6,"Filippo91","albero",False),
(7,"Mela57","frutta100",False),
(8,"LoreZZ","321progetto",False),
(9,"shadowblast01","pass777",False),
(10,"LorisZ","5G2001",False);
```

TRACCIA:

```
INSERT INTO TRACCIA("id_track","autore","versione","nome","album")
VALUES (1,"Green Day",2004,"Boulevard of Broken Dreams","American Idiot"),
(2,"Linkin Park",2000,"In The End","Hybrid Theory"),
(3,"Nino D Angelo",1982,"O studente","Nu jeans e na maglietta"),
(4,"Green Day",2004,"American Idiot","American Idiot"),
(5,"Eminem",2010,"Going Through Changes","Recovery"),
(6,"Green Day",2009,"Know Your Enemy","21st Century Breakdown"),
(7,"Articolo 31",2002,"Non è un film","Domani Smetto"),
(8,"Eminem",2010,"Not Afraid","Recovery"),
(9,"Green Day",2009,"21 Guns","21st Century Breakdown"),
(10,"Articolo 31",2002,"Gente che spera","Domani Smetto");
```

ALBUM:

```
INSERT INTO ALBUM("nome_album","artista","anno_uscita")
VALUES ("21st Century Breakdown","Green Day",2009),
("A verità","Rocco Hunt",2014),
("A verità 2.0","Rocco Hunt",2015),
("American Idiot","Green Day",2004),
("Bangarang","Skrillex",2011),
("Covered",NULL,2021),
("Covers vol.1",NULL,2020),
("DomaniSmetto","Articolo 31",2002),
("Hybrid Theory","Linkin Park",2000),
("Lace Up","Machine Gun Kelly",2012);
```

COVER:

```
INSERT INTO COVER
("autore","anno_nascita","anno_rivisitazione","nome","album","id_cover","tracci
a_originale")

VALUES

("Tommee Profitt",2000,2020,"In The End (MellenGi)","Covers vol.1",2,2),

("Sparsh Shah",2010,2016,"Not Afraid (Cover by Sparsh)","Covered",3,8),

("Fracionado",2004,2011,"American Idiot (Drum Cover)","Covered",5,4);
```

PREFERITI_TRACCIA:

```
INSERT INTO PREFERITI_TRACCIA ("id_utente","id_traccia","id_preferito")

VALUES (1,5,1),

(1,8,2),

(2,3,3),

(3,7,4),

(4,1,5),

(4,2,6),

(4,4,7),

(5,5,8),

(7,6,9),

(9,6,10);
```

PREFERITI_COVER:

```
INSERT INTO PREFERITI_COVER  ("id_utente","id_cover","id_preferito")

VALUES  (1,2,1),

(1,3,2),

(2,2,4),

(2,3,5),

(3,3,7);
```

ASCOLTO_TRACCIA:

```
INSERT INTO ASCOLTO_TRACCIA
("id_ascolto","id_utente","id_traccia","num_ascolti","fascia_oraria")
VALUES (1,1,1,20,"00:00 - 06:00"),
(2,1,1,11,"18:00 - 00:00"),
(3,1,5,10,"12:00 - 18:00"),
(4,1,10,22,"18:00 - 00:00"),
(5,2,7,21,"18:00 - 00:00"),
(6,4,4,13,"06:00 - 12:00"),
(7,5,9,18,"06:00 - 12:00"),
(8,6,8,15,"00:00 - 06:00"),
(9,8,2,12,"12:00 - 18:00"),
(10,9,3,13,"12:00 - 18:00");
```

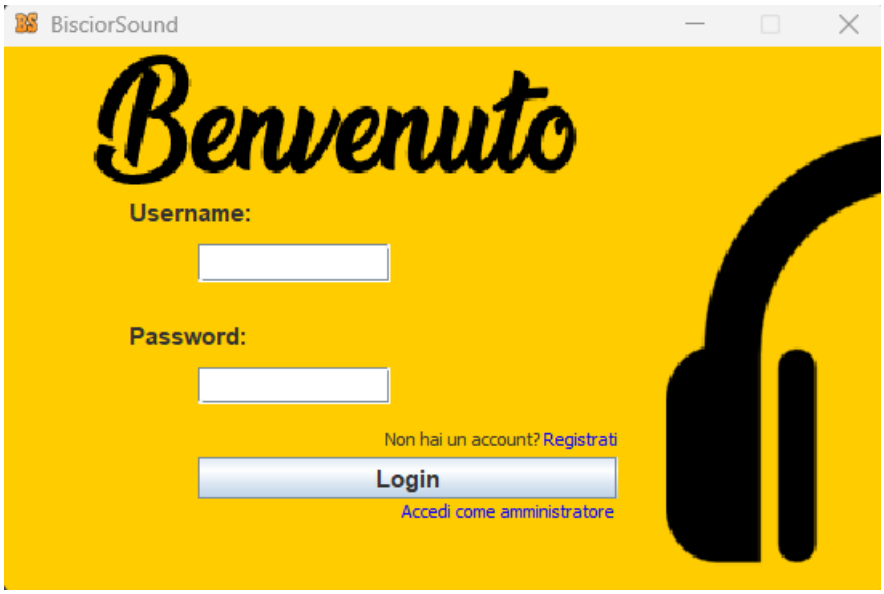
ASCOLTO_COVER:

```
INSERT INTO ASCOLTO_COVER
("id_ascoltoc","id_utente","id_cover","num_ascolti","fascia_oraria")
VALUES (1,1,5,9,"18:00 - 00:00"),
(2,2,3,15,"00:00 - 06:00"),
(3,3,2,16,"12:00 - 18:00"),
(4,6,2,10,"18:00 - 00:00"),
(5,7,3,17,"06:00 - 12:00"),
(6,9,5,11,"12:00 - 18:00");
```

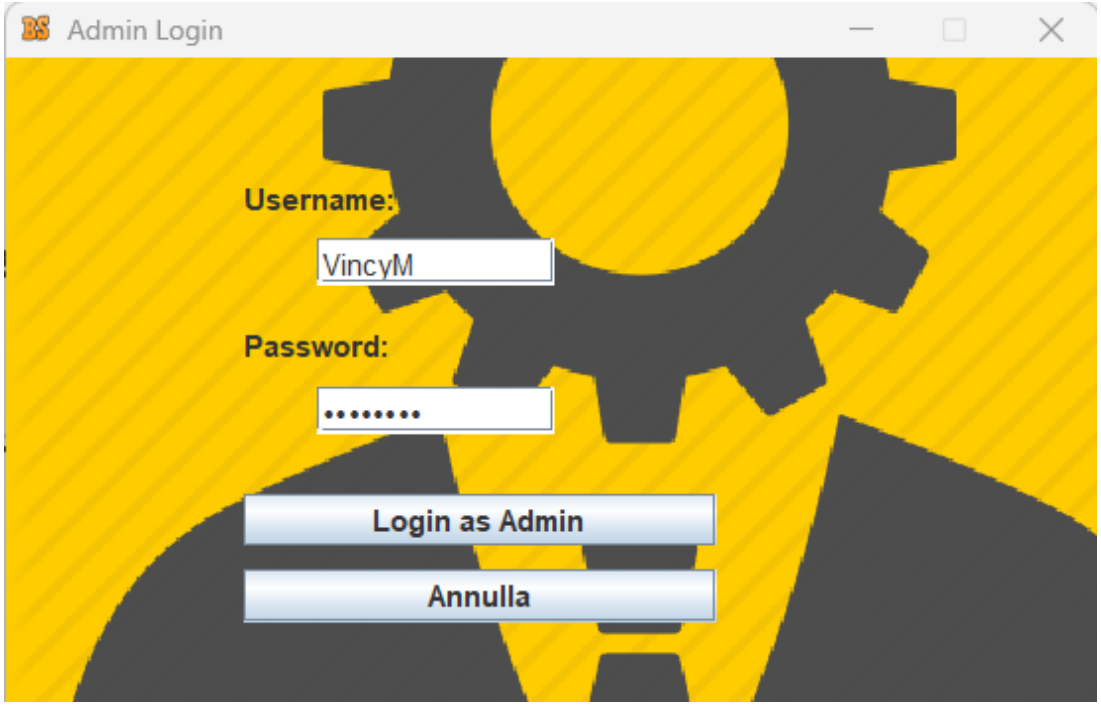
5.2 Esempio d'Uso

5.2.1 Log-in Amministratore

Per accedere come adminè necessario cliccare la sezione "Accedi come amministratore"

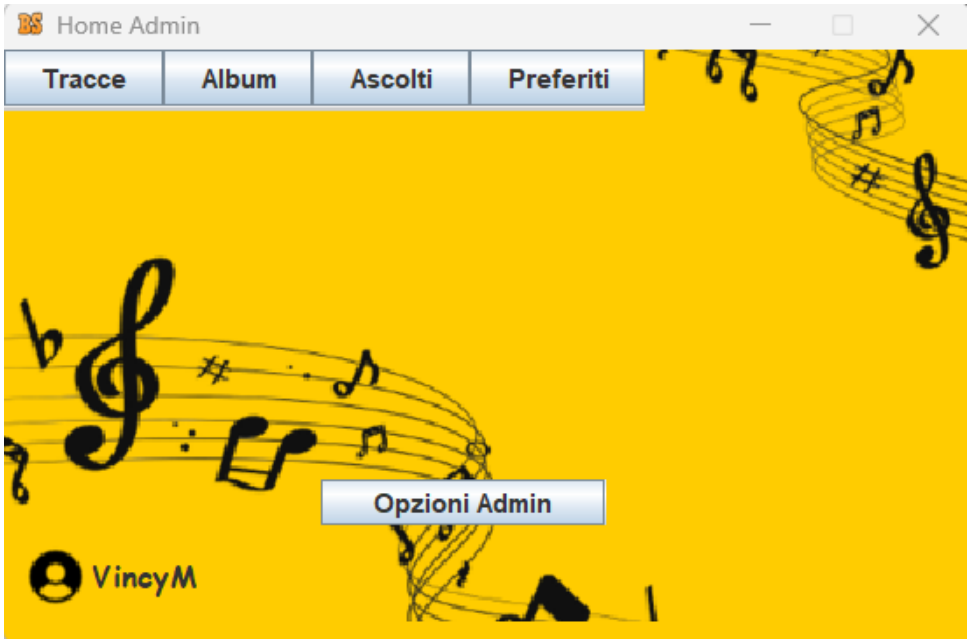


Si aprirà una nuova schermata dove l'admin avrà la possibilità di accesso, dopo aver digitato le proprie credenziali e pigiando**login as admin**,se quest'ultime saranno corrette, si accederà alla home riservata agli admin.



5.2.2 Fascia Oraria in cui l'Utente ha effettuato più ascolti

Nella home admin, c'è una sezione riservata agli admin chiamata **opzioni admin**.



Aprendo le opzioni,l'admin ha la possibilità di monitorare sia gli utenti che hanno effettuato più ascolti che la fascia oraria in cui un utente ha effettuato più ascolti.

Digitando l'username dell'utente, si aprirà una schermata dove l'admin avrà la possibilità di visionare il picco massimo della fascia oraria in cui l'utente ha effettuato più ascolti. Differenziata per tracce e per cover.