1. Abstract

MusicaViva è un sistema di gestione per uno store musicale digitale che permette l'acquisto, lo streaming e la gestione di contenuti musicali. Il database è progettato per gestire artisti, album, brani, generi musicali, utenti e transazioni. La piattaforma consente agli utenti di creare playlist personalizzate, acquistare musica in diversi formati (digitale, vinile, CD), abbonarsi a piani premium e ricevere consigli personalizzati basati sulle loro preferenze d'ascolto. MusicaViva supporta inoltre funzionalità sociali come condivisione di playlist e recensioni, facilitando l'interazione tra utenti appassionati di musica. Il sistema è progettato per gestire efficacemente grandi volumi di dati musicali, garantendo prestazioni ottimali nell'interrogazione e nell'aggiornamento delle informazioni.

2. Analisi dei Requisiti

2.1 Descrizione Testuale dei Requisiti

Il database di MusicaViva deve gestire le seguenti entità e relazioni:

Artisti: Ogni artista è caratterizzato da un identificativo univoco, nome d'arte, nome reale, data di nascita, nazionalità, biografia e immagine di profilo. Gli artisti possono essere solisti o gruppi musicali (band). Nel caso di band, si tiene traccia dei membri che la compongono con i rispettivi ruoli.

Album: Ogni album ha un identificativo univoco, titolo, data di pubblicazione, etichetta discografica, copertina, prezzo di acquisto (in diversi formati) e numero di tracce. Un album è pubblicato da uno o più artisti (collaborazioni) e appartiene a uno o più generi musicali.

Brani: Ciascun brano ha un identificativo univoco, titolo, durata, testo (se disponibile), anno di pubblicazione, dimensione del file (per il download), prezzo (se acquistabile singolarmente) e numero di riproduzioni. Ogni brano appartiene ad almeno un album, è pubblicato da almeno un artista, ed è classificato in uno o più generi musicali.

Utenti: Gli utenti del sistema sono identificati da un ID univoco e possiedono informazioni personali come nome, cognome, email, password, data di registrazione, e dati di pagamento. Gli utenti possono essere standard o premium (con abbonamento).

Playlist: Le playlist sono collezioni di brani create dagli utenti. Possiedono un identificativo univoco, un titolo, una descrizione, una data di creazione e un'impostazione di visibilità (pubblica o privata).

Transazioni: Il sistema tiene traccia di tutte le transazioni effettuate, includendo acquisti di brani/album, abbonamenti e gift card. Ogni transazione ha un ID, data, importo, stato del pagamento e metodo di pagamento.

Ascolti: Il sistema registra tutti gli ascolti degli utenti, salvando utente, brano, data/ora dell'ascolto e la durata dell'ascolto (per determinare se è stato completato).

Recensioni: Gli utenti possono lasciare recensioni su album e brani, con un punteggio da 1 a 5, un commento e una data di pubblicazione.

2.2 Operazioni Tipiche

Operazione	Tipo	Frequenza
Ricerca di brani/album/artisti	L	50.000/giorno
Riproduzione di un brano	L/S	1.000.000/giorno
Creazione/modifica playlist	S	10.000/giorno
Aggiunta di un brano a una playlist	S	50.000/giorno
Acquisto di un brano/album	S	5.000/giorno
Registrazione di un nuovo utente	S	1.000/giorno
Aggiornamento catalogo musicale	S	500/settimana
Generazione di consigli personalizzati	L	100.000/giorno
Pubblicazione di una recensione	S	2.000/giorno
Calcolo delle royalties per gli artisti	L	1/giorno
Analisi delle tendenze di ascolto	L	10/giorno
Visualizzazione cronologia ascolti	L	20.000/giorno

2.3 Glossario dei Termini

Termine	Descrizione	Collegamenti
Artista	Musicista o gruppo musicale che produce contenuti audio	Album, Brano, Band
Album	Collezione di brani pubblicata come un'unica opera	Artista, Brano, Genere
Brano	Singola composizione musicale	Album, Artista, Genere, Ascolto
Utente	Persona registrata nel sistema che può ascoltare/acquistare musica	Ascolto, Playlist, Recensione, Transazione
Utente Premium	Utente con accesso a funzionalità aggiuntive tramite abbonamento	Abbonamento
Playlist	Raccolta personalizzata di brani creata da un utente	Utente, Brano

Termine	Descrizione	Collegamenti
Transazione	Registrazione di un acquisto o abbonamento	Utente, Album, Brano, Abbonamento
Ascolto	Registrazione di una riproduzione di un brano	Utente, Brano
Genere	Categoria stilistica musicale	Album, Brano
Recensione	Valutazione e commento su un album o brano	Utente, Album, Brano
Abbonamento	Servizio premium a pagamento	Utente Premium
Band	Gruppo musicale composto da più persone	Artista, Membro Band

3. Progettazione Concettuale

3.1 Schema Concettuale (E-R)

Lo schema E-R includerebbe le seguenti entità e relazioni:

Entità:

- Artista (con generalizzazione in Solista e Band)
- Album
- Brano
- Utente (con generalizzazione in Utente Standard e Utente Premium)
- Playlist
- Transazione
- Genere Musicale
- Recensione
- Ascolto
- Membro Band (per i componenti di un gruppo musicale)
- Abbonamento

Relazioni principali:

- Pubblicazione (N:N tra Artista e Album)
- Produzione (N:N tra Artista e Brano)
- Appartenenza (N:1 tra Brano e Album)
- Classificazione (N:N tra Brano e Genere)
- Categorizzazione (N:N tra Album e Genere)
- Creazione (1:N tra Utente e Playlist)
- Composizione (N:N tra Playlist e Brano)

- Acquisto (N:N tra Utente e Brano/Album tramite Transazione)
- Riproduzione (N:N tra Utente e Brano tramite Ascolto)
- Valutazione (N:N tra Utente e Album/Brano tramite Recensione)
- Sottoscrizione (1:1 tra Utente Premium e Abbonamento)
- Appartenenza (N:1 tra Membro Band e Band)

3.2 Glossario delle Entità e Relazioni

Artista:

- ID_Artista (PK): intero, identificativo univoco
- Nome_Arte: varchar(100), nome d'arte dell'artista
- Nome_Reale: varchar(100), nome anagrafico (può essere NULL per le band)
- Data_Nascita: date, data di nascita o formazione
- Nazionalità: varchar(50), paese d'origine
- Biografia: text, descrizione della carriera
- Immagine: varchar(255), URL dell'immagine di profilo
- Tipo: enum('Solista', 'Band'), tipo di artista

Album:

- ID_Album (PK): intero, identificativo univoco
- Titolo: varchar(100), titolo dell'album
- Data_Pubblicazione: date, data di rilascio
- Etichetta: varchar(100), casa discografica
- Copertina: varchar(255), URL dell'immagine di copertina
- Prezzo_Digitale: decimal(5,2), prezzo in formato digitale
- Prezzo_CD: decimal(5,2), prezzo in formato CD (può essere NULL)
- Prezzo_Vinile: decimal(5,2), prezzo in formato vinile (può essere NULL)
- Num_Tracce: integer, numero di brani nell'album

Brano:

- ID_Brano (PK): intero, identificativo univoco
- Titolo: varchar(100), titolo del brano
- Durata: time, durata del brano
- Testo: text, testo della canzone (può essere NULL)
- Anno: integer, anno di pubblicazione
- Dimensione_File: integer, dimensione in KB
- Prezzo: decimal(4,2), prezzo per l'acquisto singolo
- Numero_Riproduzioni: integer, contatore degli ascolti totali
- Posizione_Album: integer, numero di traccia nell'album

Utente:

- ID_Utente (PK): intero, identificativo univoco
- Nome: varchar(50), nome dell'utente
- Cognome: varchar(50), cognome dell'utente
- Email: varchar(100), indirizzo email (unique)
- Password: varchar(255), password criptata
- Data_Registrazione: date, data di iscrizione
- Tipo: enum('Standard', 'Premium'), tipo di account

Utente_Premium (specializzazione di Utente):

- ID_Utente (PK, FK): intero, riferimento a Utente
- Data_Inizio_Premium: date, inizio abbonamento
- Data_Fine_Premium: date, scadenza abbonamento (può essere NULL)
- Tipo_Abbonamento: enum('Mensile', 'Annuale', 'Familiare')

Playlist:

- ID_Playlist (PK): intero, identificativo univoco
- Titolo: varchar(100), titolo della playlist
- Descrizione: text, descrizione della playlist (può essere NULL)
- Data_Creazione: datetime, data e ora di creazione
- Visibilità: enum('Pubblica', 'Privata'), impostazione di privacy
- ID_Utente (FK): intero, creatore della playlist

Transazione:

- ID_Transazione (PK): intero, identificativo univoco
- Data: datetime, data e ora della transazione
- Importo: decimal(6,2), importo totale
- Stato: enum('Completata', 'In elaborazione', 'Annullata', 'Fallita')
- Metodo_Pagamento: enum('Carta di credito', 'PayPal', 'Gift Card', 'Bonifico')
- ID_Utente (FK): intero, utente che ha effettuato la transazione

Genere_Musicale:

- ID_Genere (PK): intero, identificativo univoco
- Nome: varchar(50), nome del genere
- Descrizione: text, descrizione del genere (può essere NULL)

Ascolto:

- ID_Ascolto (PK): intero, identificativo univoco

- Data_Ora: datetime, momento dell'ascolto
- Durata_Effettiva: time, quanto a lungo l'utente ha ascoltato
- Completato: boolean, se il brano è stato ascoltato interamente
- ID_Utente (FK): intero, utente che ha ascoltato
- ID_Brano (FK): intero, brano ascoltato

Recensione:

- ID_Recensione (PK): intero, identificativo univoco
- Punteggio: integer(1-5), valutazione numerica
- Commento: text, testo della recensione (può essere NULL)
- Data: datetime, data e ora di pubblicazione
- ID_Utente (FK): intero, autore della recensione
- Oggetto_ID: intero, ID dell'album o brano recensito
- Tipo_Oggetto: enum('Album', 'Brano'), tipo di oggetto recensito

Membro_Band:

- ID_Membro (PK): intero, identificativo univoco
- Nome: varchar(100), nome del membro
- Ruolo: varchar(50), ruolo nel gruppo (es. "chitarrista", "vocalist")
- Data_Ingresso: date, quando è entrato nella band
- Data_Uscita: date, quando ha lasciato la band (può essere NULL)
- ID_Band (FK): intero, riferimento alla band di appartenenza

4. Progettazione Logica

4.1 Analisi delle Ridondanze

Una ridondanza significativa è il campo `Numero_Riproduzioni` nella tabella `Brano`. Questo valore potrebbe essere calcolato contando tutte le tuple nella tabella `Ascolto` che fanno riferimento al brano specifico.

Analisi dei volumi:

- Brani: circa 5 milioni di tuple
- Ascolti: circa 1 miliardo di tuple (con crescita costante)

Operazioni coinvolte:

1. Operazione 1: Riproduzione di un brano (aggiornamento del contatore) - 1.000.000/giorno
2. Operazione 2: Visualizzazione delle statistiche di ascolto di un brano - 50.000/giorno

Con ridondanza:

- Operazione 1:
  - Accesso in lettura a Brano: 1
  - Aggiornamento di Brano: 1
  - Inserimento in Ascolto: 1
  - Costo totale: 1L + 2S = 1.000.000L + 2.000.000S = 5.000.000 accessi/giorno
- Operazione 2:
  - Accesso in lettura a Brano: 1
  - Costo totale: 1L = 50.000L = 50.000 accessi/giorno

Costo totale con ridondanza: 5.050.000 accessi/giorno

Senza ridondanza:

- Operazione 1:
  - Inserimento in Ascolto: 1
  - Costo totale: 1S = 1.000.000S = 2.000.000 accessi/giorno
- Operazione 2:
  - Join tra Brano e Ascolto con count: circa 200L in media (per brano)
  - Costo totale: 200L * 50.000 = 10.000.000 accessi/giorno

Costo totale senza ridondanza: 12.000.000 accessi/giorno

La ridondanza comporta un risparmio di circa 7 milioni di accessi al giorno, con un costo in termini di spazio di 4 byte * 5 milioni = 20 MB circa.

Conclusione: Mantenere la ridondanza è conveniente. Il campo `Numero_Riproduzioni` sarà aggiornato ad ogni nuovo ascolto.

4.2 Eliminazione delle Generalizzazioni

4.2.1 Generalizzazione Artista (Solista/Band)

Adottiamo la strategia di **accorpamento dei figli nel padre** per la generalizzazione Artista, aggiungendo un attributo `Tipo` per distinguere tra solisti e band. Questa scelta è motivata dal fatto che le operazioni su entrambe le entità sono spesso contestuali e che la maggior parte degli attributi sono comuni.

4.2.2 Generalizzazione Utente (Standard/Premium)

Per la generalizzazione Utente, adottiamo la strategia di **accorpamento dei figli nel padre**, aggiungendo campi per gestire le informazioni degli utenti premium. Questa scelta è motivata dalla necessità di accedere spesso a tutti gli attributi contemporaneamente e di semplificare le query che coinvolgono tutti gli utenti.

4.3 Scelta degli Identificatori Primari

Si è scelto di utilizzare identificatori artificiali (auto-incrementanti) per tutte le entità principali, in modo da semplificare i riferimenti e migliorare le prestazioni delle join. Questi identificatori sono di tipo intero.

4.4 Schema Relazionale

```
Artista(ID_Artista, Nome_Arte, Nome_Reale, Data_Nascita, Nazionalità,
Biografia, Immagine, Tipo)

Membro_Band(ID_Membro, Nome, Ruolo, Data_Ingresso, Data_Uscita, ID_Band)
FK: ID_Band → Artista.ID_Artista

Album(ID_Album, Titolo, Data_Pubblicazione, Etichetta, Copertina,
Prezzo_Digitale, Prezzo_CD, Prezzo_Vinile, Num_Tracce)

Brano(ID_Brano, Titolo, Durata, Testo, Anno, Dimensione_File, Prezzo,
Numero_Riproduzioni, ID_Album, Posizione_Album)
FK: ID_Album → Album.ID_Album

Genere_Musicale(ID_Genere, Nome, Descrizione)

Brano_Genere(ID_Brano, ID_Genere)
FK: ID_Brano → Brano.ID_Brano
FK: ID_Genere → Genere_Musicale.ID_Genere

Album_Genere(ID_Album, ID_Genere)
FK: ID_Album → Album.ID_Album
FK: ID_Genere → Genere_Musicale.ID_Genere

Artista_Album(ID_Artista, ID_Album, Ruolo)
FK: ID_Artista → Artista.ID_Artista
FK: ID_Album → Album.ID_Album

Artista_Brano(ID_Artista, ID_Brano, Ruolo)
FK: ID_Artista → Artista.ID_Artista
FK: ID_Brano → Brano.ID_Brano

Utente(ID_Utente, Nome, Cognome, Email, Password, Data_Registrazione, Tipo,
Data_Inizio_Premium, Data_Fine_Premium, Tipo_Abbonamento)

Playlist(ID_Playlist, Titolo, Descrizione, Data_Creazione, Visibilità,
ID_Utente)
FK: ID_Utente → Utente.ID_Utente
```

```
Playlist_Brano(ID_Playlist, ID_Brano, Posizione)
FK: ID_Playlist → Playlist.ID_Playlist
FK: ID_Brano → Brano.ID_Brano

Transazione(ID_Transazione, Data, Importo, Stato, Metodo_Pagamento,
ID_Utente)
FK: ID_Utente → Utente.ID_Utente

Dettaglio_Transazione(ID_Transazione, Tipo_Oggetto, ID_Oggetto,
Prezzo_Unitario, Quantità)
FK: ID_Transazione → Transazione.ID_Transazione
[Tipo_Oggetto può essere 'Album', 'Brano', 'Abbonamento']

Ascolto(ID_Ascolto, Data_Ora, Durata_Effettiva, Completato, ID_Utente,
ID_Brano)
FK: ID_Utente → Utente.ID_Utente
FK: ID_Brano → Brano.ID_Brano

Recensione(ID_Recensione, Punteggio, Commento, Data, ID_Utente, Oggetto_ID,
Tipo_Oggetto)
FK: ID_Utente → Utente.ID_Utente
[Tipo_Oggetto può essere 'Album' o 'Brano']
```

5. Implementazione dello Schema Logico

Il file SQL completo per la creazione del database includerebbe:

```sql
CREATE TABLE Artista (
    ID_Artista SERIAL PRIMARY KEY,
    Nome_Arte VARCHAR(100) NOT NULL,
    Nome_Reale VARCHAR(100),
    Data_Nascita DATE,
    Nazionalità VARCHAR(50),
    Biografia TEXT,
    Immagine VARCHAR(255),
    Tipo ENUM('Solista', 'Band') NOT NULL
);

CREATE TABLE Membro_Band (
    ID_Membro SERIAL PRIMARY KEY,
    Nome VARCHAR(100) NOT NULL,
    Ruolo VARCHAR(50) NOT NULL,
    Data_Ingresso DATE NOT NULL,
    Data_Uscita DATE,
```

```
    ID_Band INT NOT NULL,
    FOREIGN KEY (ID_Band) REFERENCES Artista(ID_Artista)
);

CREATE TABLE Album (
    ID_Album SERIAL PRIMARY KEY,
    Titolo VARCHAR(100) NOT NULL,
    Data_Pubblicazione DATE NOT NULL,
    Etichetta VARCHAR(100),
    Copertina VARCHAR(255),
    Prezzo_Digitale DECIMAL(5,2),
    Prezzo_CD DECIMAL(5,2),
    Prezzo_Vinile DECIMAL(5,2),
    Num_Tracce INT NOT NULL
);

CREATE TABLE Genere_Musicale (
    ID_Genere SERIAL PRIMARY KEY,
    Nome VARCHAR(50) NOT NULL UNIQUE,
    Descrizione TEXT
);

CREATE TABLE Brano (
    ID_Brano SERIAL PRIMARY KEY,
    Titolo VARCHAR(100) NOT NULL,
    Durata TIME NOT NULL,
    Testo TEXT,
    Anno INT,
    Dimensione_File INT,
    Prezzo DECIMAL(4,2),
    Numero_Riproduzioni INT DEFAULT 0,
    ID_Album INT NOT NULL,
    Posizione_Album INT,
    FOREIGN KEY (ID_Album) REFERENCES Album(ID_Album)
);

CREATE TABLE Brano_Genere (
    ID_Brano INT NOT NULL,
    ID_Genere INT NOT NULL,
    PRIMARY KEY (ID_Brano, ID_Genere),
    FOREIGN KEY (ID_Brano) REFERENCES Brano(ID_Brano),
    FOREIGN KEY (ID_Genere) REFERENCES Genere_Musicale(ID_Genere)
);

CREATE TABLE Album_Genere (
    ID_Album INT NOT NULL,
    ID_Genere INT NOT NULL,
    PRIMARY KEY (ID_Album, ID_Genere),
    FOREIGN KEY (ID_Album) REFERENCES Album(ID_Album),
    FOREIGN KEY (ID_Genere) REFERENCES Genere_Musicale(ID_Genere)
```

```
);

CREATE TABLE Artista_Album (
    ID_Artista INT NOT NULL,
    ID_Album INT NOT NULL,
    Ruolo VARCHAR(50),
    PRIMARY KEY (ID_Artista, ID_Album),
    FOREIGN KEY (ID_Artista) REFERENCES Artista(ID_Artista),
    FOREIGN KEY (ID_Album) REFERENCES Album(ID_Album)
);

CREATE TABLE Artista_Brano (
    ID_Artista INT NOT NULL,
    ID_Brano INT NOT NULL,
    Ruolo VARCHAR(50),
    PRIMARY KEY (ID_Artista, ID_Brano),
    FOREIGN KEY (ID_Artista) REFERENCES Artista(ID_Artista),
    FOREIGN KEY (ID_Brano) REFERENCES Brano(ID_Brano)
);

CREATE TABLE Utente (
    ID_Utente SERIAL PRIMARY KEY,
    Nome VARCHAR(50) NOT NULL,
    Cognome VARCHAR(50) NOT NULL,
    Email VARCHAR(100) NOT NULL UNIQUE,
    Password VARCHAR(255) NOT NULL,
    Data_Registrazione DATE NOT NULL,
    Tipo ENUM('Standard', 'Premium') NOT NULL DEFAULT 'Standard',
    Data_Inizio_Premium DATE,
    Data_Fine_Premium DATE,
    Tipo_Abbonamento ENUM('Mensile', 'Annuale', 'Familiare')
);

CREATE TABLE Playlist (
    ID_Playlist SERIAL PRIMARY KEY,
    Titolo VARCHAR(100) NOT NULL,
    Descrizione TEXT,
    Data_Creazione TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    Visibilità ENUM('Pubblica', 'Privata') NOT NULL DEFAULT 'Privata',
    ID_Utente INT NOT NULL,
    FOREIGN KEY (ID_Utente) REFERENCES Utente(ID_Utente)
);

CREATE TABLE Playlist_Brano (
    ID_Playlist INT NOT NULL,
    ID_Brano INT NOT NULL,
    Posizione INT NOT NULL,
    PRIMARY KEY (ID_Playlist, ID_Brano),
    FOREIGN KEY (ID_Playlist) REFERENCES Playlist(ID_Playlist),
    FOREIGN KEY (ID_Brano) REFERENCES Brano(ID_Brano)
```

```
);

CREATE TABLE Transazione (
    ID_Transazione SERIAL PRIMARY KEY,
    Data TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    Importo DECIMAL(6,2) NOT NULL,
    Stato ENUM('Completata', 'In elaborazione', 'Annullata', 'Fallita') NOT
NULL,
    Metodo_Pagamento ENUM('Carta di credito', 'PayPal', 'Gift Card',
'Bonifico') NOT NULL,
    ID_Utente INT NOT NULL,
    FOREIGN KEY (ID_Utente) REFERENCES Utente(ID_Utente)
);

CREATE TABLE Dettaglio_Transazione (
    ID_Transazione INT NOT NULL,
    Tipo_Oggetto ENUM('Album', 'Brano', 'Abbonamento') NOT NULL,
    ID_Oggetto INT NOT NULL,
    Prezzo_Unitario DECIMAL(5,2) NOT NULL,
    Quantità INT NOT NULL DEFAULT 1,
    PRIMARY KEY (ID_Transazione, Tipo_Oggetto, ID_Oggetto),
    FOREIGN KEY (ID_Transazione) REFERENCES Transazione(ID_Transazione)
);

CREATE TABLE Ascolto (
    ID_Ascolto SERIAL PRIMARY KEY,
    Data_Ora TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    Durata_Effettiva TIME,
    Completato BOOLEAN NOT NULL DEFAULT FALSE,
    ID_Utente INT NOT NULL,
    ID_Brano INT NOT NULL,
    FOREIGN KEY (ID_Utente) REFERENCES Utente(ID_Utente),
    FOREIGN KEY (ID_Brano) REFERENCES Brano(ID_Brano)
);

CREATE TABLE Recensione (
    ID_Recensione SERIAL PRIMARY KEY,
    Punteggio INT NOT NULL CHECK (Punteggio BETWEEN 1 AND 5),
    Commento TEXT,
    Data TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    ID_Utente INT NOT NULL,
    Oggetto_ID INT NOT NULL,
    Tipo_Oggetto ENUM('Album', 'Brano') NOT NULL,
    FOREIGN KEY (ID_Utente) REFERENCES Utente(ID_Utente)
);

-- Trigger per aggiornare Numero_Riproduzioni quando viene inserito un nuovo
ascolto
CREATE OR REPLACE FUNCTION aggiorna_numero_riproduzioni()
RETURNS TRIGGER AS $$
```

```
BEGIN
    UPDATE Brano
    SET Numero_Riproduzioni = Numero_Riproduzioni + 1
    WHERE ID_Brano = NEW.ID_Brano;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER after_ascolto_insert
AFTER INSERT ON Ascolto
FOR EACH ROW
EXECUTE FUNCTION aggiorna_numero_riproduzioni();
```

6. Definizione delle Query e degli Indici Associati

6.1 Query

Query 1: Top 10 artisti più ascoltati negli ultimi 30 giorni

```
SELECT a.ID_Artista, a.Nome_Arte, COUNT(*) AS Numero_Ascolti
FROM Artista a
JOIN Artista_Brano ab ON a.ID_Artista = ab.ID_Artista
JOIN Brano b ON ab.ID_Brano = b.ID_Brano
JOIN Ascolto asc ON b.ID_Brano = asc.ID_Brano
WHERE asc.Data_Ora >= CURRENT_DATE - INTERVAL '30 days'
GROUP BY a.ID_Artista, a.Nome_Arte
ORDER BY Numero_Ascolti DESC
LIMIT 10;
```

Query 2: Ricavi mensili per tipo di prodotto (album, brani, abbonamenti) nell'ultimo anno

```
SELECT
    DATE_TRUNC('month', t.Data) AS Mese,
    dt.Tipo_Oggetto,
    SUM(dt.Prezzo_Unitario * dt.Quantità) AS Ricavo_Totale
FROM Transazione t
JOIN Dettaglio_Transazione dt ON t.ID_Transazione = dt.ID_Transazione
WHERE t.Stato = 'Completata'
  AND t.Data >= CURRENT_DATE - INTERVAL '1 year'
GROUP BY Mese, dt.Tipo_Oggetto
ORDER BY Mese DESC, Ricavo_Totale DESC;
```

Query 3: Brani più popolari in una playlist pubblica, raggruppati per genere

```sql
SELECT
    gm.Nome AS Genere,
    b.Titolo AS Brano,
    COUNT(pb.ID_Playlist) AS Numero_Playlist,
    SUM(b.Numero_Riproduzioni) AS Riproduzioni_Totali
FROM Brano b
JOIN Playlist_Brano pb ON b.ID_Brano = pb.ID_Brano
JOIN Playlist p ON pb.ID_Playlist = p.ID_Playlist
JOIN Brano_Genere bg ON b.ID_Brano = bg.ID_Brano
JOIN Genere_Musicale gm ON bg.ID_Genere = gm.ID_Genere
WHERE p.Visibilità = 'Pubblica'
GROUP BY gm.Nome, b.Titolo
HAVING COUNT(pb.ID_Playlist) > 5
ORDER BY gm.Nome, Numero_Playlist DESC;
```

Query 4: Album con valutazione media superiore a 4, con almeno 10 recensioni

```sql
SELECT
    a.ID_Album,
    a.Titolo,
    ar.Nome_Arte AS Artista,
    AVG(r.Punteggio) AS Valutazione_Media,
    COUNT(r.ID_Recensione) AS Numero_Recensioni
FROM Album a
JOIN Artista_Album aa ON a.ID_Album = aa.ID_Album
JOIN Artista ar ON aa.ID_Artista = ar.ID_Artista
JOIN Recensione r ON a.ID_Album = r.Oggetto_ID AND r.Tipo_Oggetto = 'Album'
GROUP BY a.ID_Album, a.Titolo, ar.Nome_Arte
HAVING AVG(r.Punteggio) > 4 AND COUNT(r.ID_Recensione) >= 10
ORDER BY Valutazione_Media DESC;
```

Query 5: Per ogni utente premium, mostrare il numero di brani ascoltati, il tempo totale di ascolto e il risparmio effettuato (confronto con acquisto dei brani)

```sql
SELECT
    u.ID_Utente,
    CONCAT(u.Nome, ' ', u.Cognome) AS Nome_Completo,
    COUNT(DISTINCT a.ID_Brano) AS Brani_Diversi_Ascoltati,
    SUM(EXTRACT(EPOCH FROM a.Durata_Effettiva))/3600 AS Ore_Totali_Ascolto,
    SUM(b.Prezzo) AS Valore_Brani_Ascoltati,
    CASE
        WHEN u.Tipo_Abbonamento = 'Mensile' THEN COUNT(DISTINCT
DATE_TRUNC('month', a.Data_Ora)) * 9.99
        WHEN u.Tipo_Abbonamento = 'Annuale' THEN COUNT(DISTINCT
```

```
DATE_TRUNC('year', a.Data_Ora)) * 99.99
        WHEN u.Tipo_Abbonamento = 'Familiare' THEN COUNT(DISTINCT
DATE_TRUNC('month', a.Data_Ora)) * 14.99
    END AS Costo_Abbonamento,
    SUM(b.Prezzo) -
    CASE
        WHEN u.Tipo_Abbonamento = 'Mensile' THEN COUNT(DISTINCT
DATE_TRUNC('month', a.Data_Ora)) * 9.99
        WHEN u.Tipo_Abbonamento = 'Annuale' THEN COUNT(DISTINCT
DATE_TRUNC('year', a.Data_Ora)) * 99.99
        WHEN u.Tipo_Abbonamento = 'Familiare' THEN COUNT(DISTINCT
DATE_TRUNC('month', a.Data_Ora)) * 14.99
    END AS Risparmio
FROM Utente u
JOIN Ascolto a ON u.ID_Utente = a.ID_Utente
JOIN Brano b ON a.ID_Brano = b.ID_Brano
WHERE u.Tipo = 'Premium'
GROUP BY u.ID_Utente, u.Nome, u.Cognome, u.Tipo_Abbonamento
HAVING COUNT(DISTINCT a.ID_Brano) > 10
ORDER BY Risparmio DESC;
```

6.2 Indici

Per ottimizzare le query sopra descritte, sono necessari i seguenti indici:

```
-- Indice per migliorare le prestazioni di ricerca nelle join tra Ascolto e
Brano
CREATE INDEX idx_ascolto_brano ON Ascolto(ID_Brano);

-- Indice per migliorare le ricerche su Data_Ora in Ascolto
CREATE INDEX idx_ascolto_data ON Ascolto(Data_Ora);

-- Indice per ottimizzare le join tra Recensione e i suoi oggetti
CREATE INDEX idx_recensione_oggetto ON Recensione(Tipo_Oggetto, Oggetto_ID);

-- Indice per migliorare le ricerche di transazioni per stato e data
CREATE INDEX idx_transazione_stato_data ON Transazione(Stato, Data);

-- Indice per migliorare le prestazioni di join tra Playlist_Brano e
Playlist
CREATE INDEX idx_playlist_brano_playlist ON Playlist_Brano(ID_Playlist);

-- Indice composito per le relazioni Artista_Brano
CREATE INDEX idx_artista_brano ON Artista_Brano(ID_Artista, ID_Brano);
```

In particolare, l'indice `idx_ascolto_data` è particolarmente significativo per la Query 1, che richiede di filtrare gli ascolti per data. Poiché questa query è eseguita frequentemente e

coinvolge un join su una grande tabella (Ascolto), l'indice ridurrà notevolmente il tempo di esecuzione.

Motivazione: La tabella Ascolto è tra le più popolate del database (milioni di righe) e la selezione per intervallo di date è un'operazione frequente. Senza questo indice, il sistema dovrebbe scansionare l'intera tabella per trovare i record degli ultimi 30 giorni, operazione estremamente costosa. Con l'indice, l'accesso ai dati sarà logaritmico invece che lineare rispetto al numero di tuple della tabella.

7. Applicazione Software

L'applicazione software che accede al database di MusicaViva implementa le query sopra definite e include le seguenti funzionalità:

1. Interfaccia di ricerca per brani, album e artisti
2. Gestione profilo utente e abbonamenti
3. Riproduzione musicale e gestione playlist
4. Sistema di transazioni e pagamenti
5. Analisi dei dati di ascolto e generazione di consigli personalizzati

Di seguito è riportato un esempio di codice C che implementa la Query 1:

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <libpq-fe.h>

void checkConnection(PGconn *conn) {
    if (PQstatus(conn) != CONNECTION_OK) {
        fprintf(stderr, "Connessione fallita: %s\n", PQerrorMessage(conn));
        PQfinish(conn);
        exit(1);
    }
}

void printQueryResults(PGresult *res) {
    int rows = PQntuples(res);
    int cols = PQnfields(res);

    // Stampa intestazioni
    for (int i = 0; i < cols; i++) {
        printf("%-25s", PQfname(res, i));
    }
    printf("\n");

    // Stampa linea di separazione
    for (int i = 0; i < cols; i++) {
```

```
        for (int j = 0; j < 25; j++) printf("-");
    }
    printf("\n");

    // Stampa dati
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            printf("%-25s", PQgetvalue(res, i, j));
        }
        printf("\n");
    }
}

int main() {
    PGconn *conn;
    PGresult *res;

    // Connessione al database
    conn = PQconnectdb("host=localhost dbname=musicaviva user=postgres
password=password");
    checkConnection(conn);

    printf("Connesso al database MusicaViva\n\n");

    // Query 1: Top 10 artisti più ascoltati negli ultimi 30 giorni
    const char *query1 =
        "SELECT a.ID_Artista, a.Nome_Arte, COUNT(*) AS Numero_Ascolti "
        "FROM Artista a "
        "JOIN Artista_Brano ab ON a.ID_Artista = ab.ID_Artista "
        "JOIN Brano b ON ab.ID_Brano = b.ID_Brano "
        "JOIN Ascolto asc ON b.ID_Brano = asc.ID_Brano "
        "WHERE asc.Data_Ora >= CURRENT_DATE - INTERVAL '30 days' "
        "GROUP BY a.ID_Artista, a.Nome_Arte "
        "ORDER BY Numero_Ascolti DESC "
        "LIMIT 10;";

    // Esecuzione query
    res = PQexec(conn, query1);

    if (PQresultStatus(res) != PGRES_TUPLES_OK) {
        fprintf(stderr, "Esecuzione della query fallita: %s\n",
PQerrorMessage(conn));
        PQclear(res);
        PQfinish(conn);
        exit(1);
    }

    printf("TOP 10 ARTISTI PIÙ ASCOLTATI NEGLI ULTIMI 30 GIORNI:\n");
    printQueryResults(res);
```

```
    // Pulizia
    PQclear(res);
    PQfinish(conn);

    return 0;
}
```

Questo codice si connette al database, esegue la Query 1 per trovare i 10 artisti più ascoltati negli ultimi 30 giorni, e stampa i risultati in forma tabellare.

L'applicazione completa implementerebbe tutte le query definite e fornirebbe un'interfaccia utente per interagire con il database in modo intuitivo, permettendo agli utenti di gestire la propria collezione musicale, creare playlist, acquistare brani/album e scoprire nuova musica.