**ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH**

**TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN**

**KHOA KHOA HỌC VÀ KỸ THUẬT THÔNG TIN**

---------



**BÁO CÁO ĐỒ ÁN**

**CƠ SỞ DỮ LIỆU CHO ỨNG DỤNG TÌM KIẾM BÀI NHẠC QUA GIAI ĐIỆU**

**Sinh viên thực hiện:**

Triệu Đức Duy - 23520392

Đỗ Trần Thế Bảo - 23520096

Nguyễn Đình Đại - 23520216

**Giảng viên:**

TS Nguyễn Gia Tuấn Anh

Th.S Phạm Nguyễn Phúc Toàn

Thành phố Hồ Chí Minh, tháng 05 năm 2025

**BÁO CÁO TÓM TẮT**

**1. Tiêu đề báo cáo: Cơ sở dữ liệu cho ứng dụng tìm kiếm bài nhạc qua giai điệu**

**2.** **Danh sách thành viên**

|  |  |  |
| --- | --- | --- |
| **MSSV** | **Họ tên** | **Ghi chú** |
| 23520392 | Triệu Đức Duy | Trưởng nhóm |
| 23520096 | Đỗ Trần Thế Bảo |  |
| 23520216 | Nguyễn Đình Đại |  |

**3.** **Nội dung chi tiết**

CHƯƠNG 1: GIỚI THIỆU BÀI TOÁN, SO SÁNH THỊ TRƯỜNG, ĐỐI TƯỢNG SỬ DỤNG, LỰA CHỌN CÔNG CỤ

1. Giới thiệu bài toán
2. So sánh thị trường
3. Đối tượng sử dụng
4. Lựa chọn công cụ

CHƯƠNG 2: PHÂN TÍCH BÀI TOÁN

1. Mô hình thực thể mối kết hợp
2. Sơ đồ ERD

CHƯƠNG 3: THIẾT KẾ CSDL

1. Phân tích yêu cầu tìm kiếm bài hát qua giai điệu:
2. Giải thích và lý do lưu trữ một số thuộc tính:
3. Bảng mô tả lược đồ quan hệ:
4. Sơ đồ quan hệ

CHƯƠNG 4: CÀI ĐẶT BẢNG, CỘT, KHÓA CHÍNH, KHÓA NGOẠI, CONSTRAINT

1. Code cài đặt
2. Giải thích một số chi tiết

CHƯƠNG 5: CÀI ĐẶT RÀNG BUỘC ĐỂ BẢO VỆ TÍNH TOÀN VẸN DỮ LIỆU

1. Các ràng buộc cần thiết cho CSDL
2. Cách để tạo các ràng buộc

CHƯƠNG 6: FUNCTION, STORED PROCED, VIEW

1. Function
2. Stored procedure
3. View

CHƯƠNG 7: PHÂN QUYỀN CƠ SỞ DỮ LIỆU

1. Phân tích
2. Code cài đặt

CHƯƠNG 8: BIỂU DIỄN THÔNG TIN

1. Form đăng nhập, đăng ký tài khoản
2. Menu các chức năng của ứng dụng
3. Report thống kê hoạt động của người dùng

CHƯƠNG 9: SAO LƯU, KHÔI PHỤC

1. Sao lưu
2. Khôi phục

CHƯƠNG 10: CÁC VẤN ĐỀ KHÁC

1. Đề xuất một cách thu thập dữ liệu cho cột đặc trưng giai điệu
2. Hướng phát triển

CHƯƠNG 11: KẾT LUẬN

1. Tự đánh giá ưu điểm
2. Tự đánh giá khuyết điểm

**4. Phân công công việc**

|  |  |  |
| --- | --- | --- |
| **MSSV** | **Họ tên** | **Nội dung được phân công** |
| 23520392 | Triệu Đức Duy | Chương 1, 2, 3, 5, 6, 8, 10, 11, Code web demo |
| 23520096 | Đỗ Trần Thế Bảo | Chương 2, 3, 4, 8 |
| 23520216 | Nguyễn Đình Đại | Chương 2, 3, 7, 9 |

**MỤC LỤC**

[CHƯƠNG 1: GIỚI THIỆU BÀI TOÁN, MỤC TIÊU, ĐỐI TƯỢNG SỬ DỤNG, LỰA CHỌN CÔNG CỤ 5](#_Toc198919923)

[1. Giới thiệu bài toán: 5](#_Toc198919924)

[2. So sánh thị trường: 5](#_Toc198919925)

[3. Đối tượng sử dụng: 6](#_Toc198919926)

[4. Lựa chọn công cụ: PostgreSQL, pgAdmin 4 vì tận dụng kiểu dữ liệu mảng sẵn có của PostgreSQL 6](#_Toc198919927)

[CHƯƠNG 2: PHÂN TÍCH BÀI TOÁN 7](#_Toc198919928)

[1. Mô hình thực thể mối kết hợp: 7](#_Toc198919929)

[2. Sơ đồ ERD: 7](#_Toc198919930)

[CHƯƠNG 3: THIẾT KẾ CSDL 8](#_Toc198919931)

[1. Phân tích yêu cầu tìm kiếm bài hát qua giai điệu: 8](#_Toc198919932)

[2. Giải thích và lý do lưu trữ một số thuộc tính: 8](#_Toc198919933)

[3. Bảng mô tả lược đồ quan hệ: 9](#_Toc198919934)

[4. Sơ đồ quan hệ 11](#_Toc198919935)

[CHƯƠNG 4: CÀI ĐẶT BẢNG, CỘT, KHÓA CHÍNH, KHÓA NGOẠI, CONSTRAINT 11](#_Toc198919936)

[1. Code cài đặt 11](#_Toc198919937)

[2. Giải thích một số chi tiết 14](#_Toc198919938)

[CHƯƠNG 5: CÀI ĐẶT RÀNG BUỘC ĐỂ BẢO VỆ TÍNH TOÀN VẸN DỮ LIỆU 14](#_Toc198919939)

[1. Các ràng buộc cần thiết cho CSDL. (Đánh số các ràng buộc ở cuối dòng để dễ gọi tên) 14](#_Toc198919940)

[2. Cách để tạo các ràng buộc 15](#_Toc198919941)

[CHƯƠNG 6: FUNCTION, STORED PROCEDURE, VIEW 21](#_Toc198919942)

[1. Function 21](#_Toc198919943)

[2. Stored procedure 22](#_Toc198919944)

[CHƯƠNG 7: PHÂN QUYỀN CƠ SỞ DỮ LIỆU 24](#_Toc198919945)

[1. Phân tích 24](#_Toc198919946)

[2. Code cài đặt 25](#_Toc198919947)

[CHƯƠNG 8: BIỂU DIỄN THÔNG TIN 25](#_Toc198919948)

[1. Form đăng nhập, đăng ký tài khoản 25](#_Toc198919949)

[Hình 8.1: Form đăng nhập 25](#_Toc198919950)

[2. Menu các chức năng của ứng dụng 25](#_Toc198919951)

[3. Report thống kê hoạt động của người dùng 25](#_Toc198919952)

[CHƯƠNG 9: SAO LƯU, KHÔI PHỤC 26](#_Toc198919953)

[1. Sao lưu 26](#_Toc198919954)

[2. Khôi phục 26](#_Toc198919955)

[CHƯƠNG 10: CÁC VẤN ĐỀ KHÁC 27](#_Toc198919956)

[1. Đề xuất một cách thu thập dữ liệu cho cột đặc trưng giai điệu 27](#_Toc198919957)

[2. Hướng phát triển 29](#_Toc198919958)

[CHƯƠNG 11: KẾT LUẬN 30](#_Toc198919959)

[1. Tự đánh giá ưu điểm 30](#_Toc198919960)

[2. Tự đánh giá khuyết điểm 30](#_Toc198919961)

**DANH MỤC BẢNG (NẾU CÓ)**

Bảng 3.1. Bảng mô tả lược đồ quan hệ 9

**DANH MỤC HÌNH VẼ (NẾU CÓ)**

Hình 2.1 Sơ đồ ERD 7

Hình 3.1. Sơ đồ quan hệ 10

Hình 8.1 Form đăng nhập 25

Hình 8.2 Menu chức năng 25

Hình 8.3 Report thống kê hoạt động 26

Hình 9.1 Giao diện sao lưu 26

Hình 9.2 Giao diện khôi phục 27

# **CHƯƠNG 1: GIỚI THIỆU BÀI TOÁN, MỤC TIÊU, ĐỐI TƯỢNG SỬ DỤNG, LỰA CHỌN CÔNG CỤ**

1. **Giới thiệu bài toán:**

* Xây dựng cơ sở dữ liệu cho ứng dụng tìm kiếm bài nhạc qua giai điệu và quản lý danh sách phát nhạc của người dùng
* Quy trình hoạt động của ứng dụng: Người dùng đăng ký tài khoản, đăng nhập vào ứng dụng, sau đó người dùng sẽ được thực hiện các chức năng như tra cứu bài nhạc, tạo danh sách phát…
* Các chức năng cần có:
* Đăng ký tài khoản, đăng nhập vào ứng dụng
* Tra cứu bài nhạc thông qua giai điệu (tiếng hát, tiếng ngân nga, âm thanh từ nhạc cụ, thiết bị,…)
* Phát bài nhạc
* Xem thông tin các bài nhạc mình tra cứu (tên bài, tác giả, ngày phát hành)
* Xem thông tin tác giả (tên, tuổi, ảnh tác giả)
* Tạo các danh sách phát nhạc
* Thêm, xóa bài nhạc trong danh sách phát
* Thay đổi thứ tự bài nhạc trong danh sách phát
* Xem lại lịch sử nhạc đã nghe

1. **So sánh thị trường:**

* So sánh thị trường:
  + Ứng dụng tìm kiếm bài nhạc qua giai điệu: Shazam
  + Ứng dụng quản lý danh sách phát: Spotify
  + Đặc điểm:
* Shazam: tìm kiếm rất tốt bài nhạc phát ra từ bản audio, không tìm kiếm được nếu người dùng ngân nga giai điệu bằng giọng hát
* Spotify: quản lý danh sách phát với nhiều tính năng, các tính năng nâng cao cần trả phí để được sử dụng, tìm kiếm bài nhạc theo tên nhưng không thể tìm theo giai điệu

1. **Đối tượng sử dụng:**

* Người muốn tìm kiếm bài nhạc nhưng không biết tên, lời bài nhạc
* Người muốn tìm kiếm bài nhạc không lời
* Người muốn tạo, tùy chỉnh các danh sách phát nhạc của mình

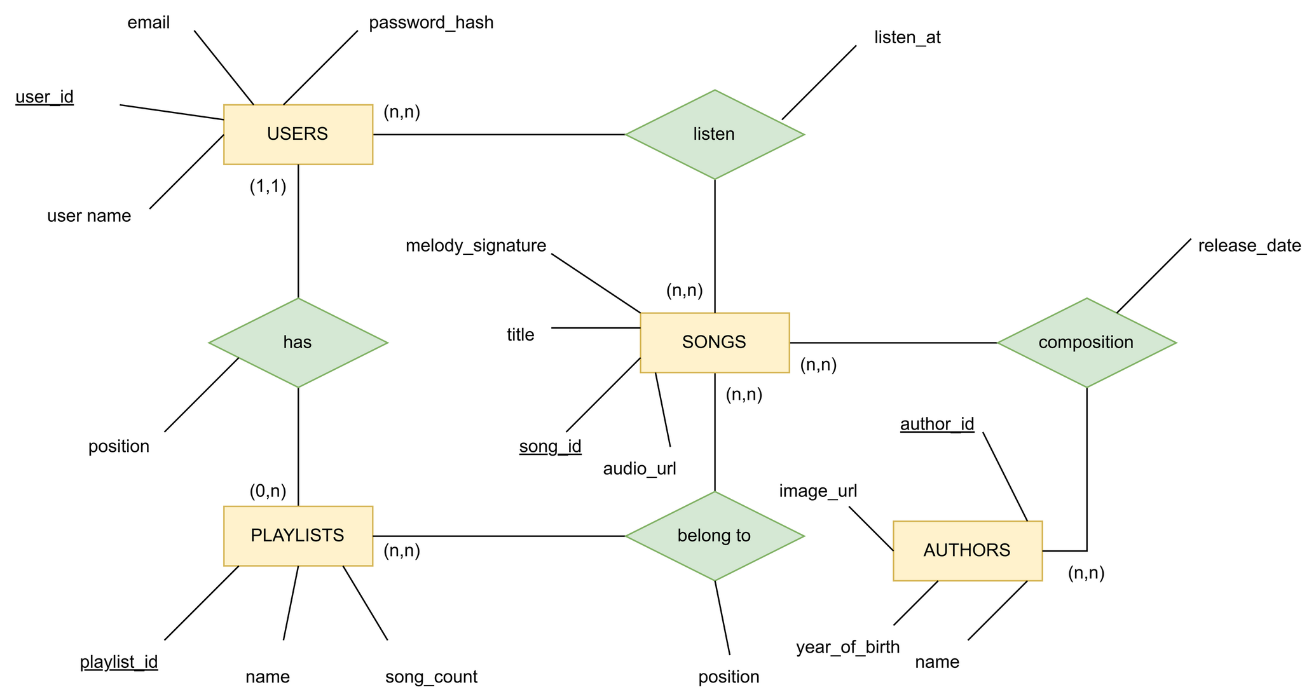
1. **Lựa chọn công cụ:** PostgreSQL, pgAdmin 4 vì tận dụng kiểu dữ liệu mảng sẵn có của PostgreSQL

# **CHƯƠNG 2: PHÂN TÍCH BÀI TOÁN**

1. **Mô hình thực thể mối kết hợp:**

* Các đối tượng cần quản lý:
* Người dùng
* Bài nhạc
* Tác giả
* Danh sách phát
* Các mối quan hệ:
* Người dùng ***nghe*** nhạc: n-n
* Tác giả ***sáng tác*** nhạc: n-n
* Người dùng ***có*** danh sách phát: 1-n
* Bài nhạc ***thuộc*** danh sách phát: n-n

1. **Sơ đồ ERD:**



Hình 2.1. Sơ đồ ERD

# **CHƯƠNG 3: THIẾT KẾ CSDL**

1. **Phân tích yêu cầu tìm kiếm bài hát qua giai điệu:**

* **Yêu cầu**: Tìm kiếm bài hát qua một đoạn giai điệu (không phải toàn bộ giai điệu của bài) ở mọi tone khác nhau vì người dùng có nam, nữ và có thể ngân nga giai điệu ở các tone khác nhau
* **Vấn đề đặt ra**: Tìm cách số hóa giai điệu bài nhạc thành dạng dữ liệu dễ lưu trữ, dễ truy vấn mà vẫn đáp ứng đủ yêu cầu ở trên
* **Kiến thức nhạc lý cần thiết để giải quyết vấn đề**: Mỗi nốt nhạc có một cao độ khác nhau, thường đo bằng Hz hoặc cung. Trong trường hợp này, lựa chọn đo bằng cung vì cung là thang đo đơn giản chỉ gồm cung và nửa cung. Ví dụ như nốt Đô thấp hơn Rê (ở cùng quãng tám) một cung, Đô thấp hơn đô thăng nửa cung, và cứ thế nốt Đô tiếp theo cũng cách nốt Rê tương tự, hai nốt Đô liền nhau cách nhau 6 cung.
* **Giải pháp cho vấn đề:** Với một đoạn nhạc, ta sẽ số hóa giai điệu của nó như sau  
  ví dụ đoạn nhạc là Đô Rê Mi Fa Sol (các nốt cùng quãng tám) thì mảng đặc trưng cho giai điệu đó sẽ là {1, 1, 0.5, 1} vì Rê cách Đô một cung, Mi cách Rê một cung, Fa cách Mi nửa cung, Sol cách Fa một cung.
* **Giải pháp**: ta cài đặt bảng songs có một cột lưu melody\_signature dưới dạng mảng các số thực (real[])
* **Vấn đề phát sinh**: việc truy vấn trên mảng khó => ta lưu thêm một cột là melody\_signature\_str = array\_to\_string(melody\_signature, ',') để dùng truy vấn LIKE ‘%...%’ để truy vấn qua một đoạn giai điệu chứ không phải toàn bộ giai điệu của bài, việc lưu melody\_signature\_str có hiệu suất tốt hơn so với việc mỗi lần truy vấn ta lại gọi hàm để tạo melody\_signature\_str từ melody\_signature

1. **Giải thích và lý do lưu trữ một số thuộc tính:**

* Password\_hash của bảng users: lưu mật khẩu băm, thuật toán băm và thực hiện băm do Backend đảm nhiệm, CSDL chỉ có nhiệm vụ lưu mật khẩu sau khi băm, nhằm đảm bảo an toàn nếu mật khẩu băm ở CSDL bị lộ thì mật khẩu gốc vẫn an toàn
* audio\_url của bảng songs: đường dẫn tới bài nhạc, giúp Backend truy cập đến file bài nhạc để thực hiện chức năng phát nhạc
* image\_url của bảng authors: đường dẫn tới ảnh tác giả, giúp xuất ảnh ra Frontend
* song\_count của bảng playlists: lưu số lượng bài nhạc trong danh sách phát, dù có thể truy vấn đếm số hàng ở playlist\_details có playlist\_id thỏa mãn để biết số lượng bài nhạc trong danh sách phát nào đó, nhưng vì không muốn mỗi lần như vậy đều phải chạy lại câu truy vấn, kết bảng, so sánh… nên lưu song\_count để tang hiệu suất, đổi lại, ta chấp nhận viết ràng buộc để đảm bảo song\_count không bị sai khi bảng playlist\_details có thêm hoặc xóa
* position của bảng playlist\_details: thứ tự trong danh sách phát của bài nhạc

1. **Bảng mô tả lược đồ quan hệ:**

|  |  |  |  |
| --- | --- | --- | --- |
| Quan hệ | Thuộc tính | Kiểu dữ liệu | Diễn giải |
| users | user\_id | serial | Khóa chính |
| username | character varying(100) | Not null |
| email | character varying(255) | unique |
| password\_hash | character varying(255) |  |
| songs | song\_id | serial | Khóa chính |
| title | character varying(255) | Not null |
| melody\_signature | real[] |  |
| audio\_url | character varying(2048) | unique |
| melody\_signature\_str | text | unique |
| authors | author\_id | serial | Khóa chính |
| name | character varying(255) | Not null |
| year\_of\_birth | integer |  |
| image\_url | character varying(2048) |  |
| playlists | playlist\_id | serial | Khóa chính |
| user\_id | integer | Tham chiếu user\_id |
| name | character varying(255) | Not null |
| song\_count | integer |  |
| playlist\_details | playlist\_id | integer | Khóa chính  Tham chiếu playlist\_id |
| song\_id | integer | Khóa chính  Tham chiếu song\_id |
| position | integer |  |
| compositions | song\_id | integer | Khóa chính  Tham chiếu song\_id |
| author\_id | integer | Khóa chính  Tham chiếu author\_id |
| song\_release\_date | date |  |
| listening\_history | user\_id | integer | Khóa chính  Tham chiếu user\_id |
| song\_id | integer | Khóa chính  Tham chiếu song\_id |
| listened\_at | timestamp | Khóa chính |

Bảng 3.1. Bảng mô tả lược đồ quan hệ

1. **Sơ đồ quan hệ**

**A screenshot of a computer

AI-generated content may be incorrect.**

Hình 3.1. Sơ đồ quan hệ

# **CHƯƠNG 4: CÀI ĐẶT BẢNG, CỘT, KHÓA CHÍNH, KHÓA NGOẠI, CONSTRAINT**

1. **Code cài đặt**

BEGIN;

CREATE TABLE IF NOT EXISTS public.authors

(

author\_id serial NOT NULL,

name character varying(255) COLLATE pg\_catalog."default" NOT NULL,

year\_of\_birth integer,

image\_url character varying(2048) COLLATE pg\_catalog."default",

CONSTRAINT authors\_pkey PRIMARY KEY (author\_id)

);

CREATE TABLE IF NOT EXISTS public.compositions

(

song\_id integer NOT NULL,

author\_id integer NOT NULL,

song\_release\_date date,

CONSTRAINT compositions\_pkey PRIMARY KEY (song\_id, author\_id)

);

CREATE TABLE IF NOT EXISTS public.listening\_history

(

user\_id integer NOT NULL,

song\_id integer NOT NULL,

listened\_at timestamp without time zone NOT NULL DEFAULT CURRENT\_TIMESTAMP,

CONSTRAINT listening\_history\_pkey PRIMARY KEY (user\_id, song\_id, listened\_at)

);

CREATE TABLE IF NOT EXISTS public.playlist\_details

(

playlist\_id integer NOT NULL,

song\_id integer NOT NULL,

"position" integer,

CONSTRAINT playlist\_details\_pkey PRIMARY KEY (playlist\_id, song\_id)

);

CREATE TABLE IF NOT EXISTS public.playlists

(

playlist\_id serial NOT NULL,

user\_id integer NOT NULL,

name character varying(255) COLLATE pg\_catalog."default" NOT NULL,

song\_count integer DEFAULT 0,

CONSTRAINT playlists\_pkey PRIMARY KEY (playlist\_id)

);

CREATE TABLE IF NOT EXISTS public.songs

(

song\_id serial NOT NULL,

title character varying(255) COLLATE pg\_catalog."default" NOT NULL,

melody\_signature real[] NOT NULL,

audio\_url character varying(2048) COLLATE pg\_catalog."default",

melody\_signature\_str text COLLATE pg\_catalog."default",

CONSTRAINT songs\_pkey PRIMARY KEY (song\_id),

CONSTRAINT unique\_audio\_url UNIQUE (audio\_url),

CONSTRAINT unique\_melody\_signature\_str UNIQUE (melody\_signature\_str)

);

CREATE TABLE IF NOT EXISTS public.users

(

user\_id serial NOT NULL,

username character varying(100) COLLATE pg\_catalog."default" NOT NULL,

email character varying(255) COLLATE pg\_catalog."default" NOT NULL,

password\_hash character varying(255) COLLATE pg\_catalog."default" NOT NULL,

CONSTRAINT users\_pkey PRIMARY KEY (user\_id),

CONSTRAINT users\_email\_key UNIQUE (email)

);

ALTER TABLE IF EXISTS public.compositions

ADD CONSTRAINT compositions\_author\_id\_fkey FOREIGN KEY (author\_id)

REFERENCES public.authors (author\_id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE RESTRICT;

ALTER TABLE IF EXISTS public.compositions

ADD CONSTRAINT compositions\_song\_id\_fkey FOREIGN KEY (song\_id)

REFERENCES public.songs (song\_id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE RESTRICT;

ALTER TABLE IF EXISTS public.listening\_history

ADD CONSTRAINT listening\_history\_song\_id\_fkey FOREIGN KEY (song\_id)

REFERENCES public.songs (song\_id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE RESTRICT;

ALTER TABLE IF EXISTS public.listening\_history

ADD CONSTRAINT listening\_history\_user\_id\_fkey FOREIGN KEY (user\_id)

REFERENCES public.users (user\_id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE RESTRICT;

ALTER TABLE IF EXISTS public.playlist\_details

ADD CONSTRAINT playlist\_details\_playlist\_id\_fkey FOREIGN KEY (playlist\_id)

REFERENCES public.playlists (playlist\_id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE RESTRICT;

ALTER TABLE IF EXISTS public.playlist\_details

ADD CONSTRAINT playlist\_details\_song\_id\_fkey FOREIGN KEY (song\_id)

REFERENCES public.songs (song\_id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE RESTRICT;

ALTER TABLE IF EXISTS public.playlists

ADD CONSTRAINT playlists\_user\_id\_fkey FOREIGN KEY (user\_id)

REFERENCES public.users (user\_id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE RESTRICT;

END;

1. **Giải thích một số chi tiết**

* listened\_at timestamp without time zone NOT NULL DEFAULT CURRENT\_TIMESTAMP: thuộc tính listened\_at là mốc thời gian nhưng bỏ múi giờ và được mặc định là thời gian lúc thêm hàng vào bảng
* ON UPDATE NO ACTION: để không cho phép sửa đổi thuộc tính nào có khóa ngoại tham chiếu tới
* ON DELETE RESTRICT: để không cho phép xóa hàng nào mà có khóa ngoại tham chiếu tới thuộc tính của nó

**CHƯƠNG 5: CÀI ĐẶT RÀNG BUỘC ĐỂ BẢO VỆ TÍNH TOÀN VẸN DỮ LIỆU**

1. **Các ràng buộc cần thiết cho CSDL.** (Đánh số các ràng buộc ở cuối dòng để dễ gọi tên)

- Email của người dùng là duy nhất: đã cài đặt ở trên khi tạo bảng (1)

- Url của bài hát là duy nhất: đã cài ở trên khi tạo bảng (2)

- Giai điệu của bài nhạc là duy nhất (3)

- Ở mỗi hàng trong bảng songs, thuộc tính melody\_signature\_str luôn bằng array\_to\_string(melody\_signature, ‘,’) (4)

- Danh sách phát mới tạo (chưa được thêm bài nhạc) sẽ có song\_count = 0, song\_count sẽ cập nhật lại khi bài nhạc được thêm vào danh sách hoặc xóa khỏi danh sách (5)

- Đảm bảo position của bài nhạc là đúng mỗi khi thêm, xóa, sửa(6)

- Không được sửa khóa ngoại playlist\_id và song\_id trong bảng playlist\_details (7)

- Không được sửa thuộc tính có khóa ngoại tham chiếu tới: đã cài đặt ở trên khi tạo bảng (8)

- Không được xóa hàng có khóa ngoại tham chiếu tới thuộc tính của nó: đã cài đặt ở trên khi tạo bảng (9)

## **2. Cách để tạo các ràng buộc**

- Các ràng buộc (1), (2), (8), (9) đã được cài đặt ở trên khi tạo bảng

- Cài đặt ràng buộc (3), (4):

* Nếu cài đặt unique ở thuộc tính melody\_signature có cấu trúc là array thì sẽ gây khó khăn trong quá trình so sánh để xác định tính unique từ đó làm chậm các quá trình thêm dữ liệu và giảm hiệu suất nên ta sẽ cài đặt unique cho melody\_signature\_str (đã cài ở trên), đồng thời không cho phép sửa trực tiếp melody\_signature\_str, chỉ cho thêm, sửa melody\_signature và sau đó melody\_signature\_str sẽ được tự động cập nhật theo
* Cài đặt không cho sửa trực tiếp melody\_signature\_str:

CREATE OR REPLACE FUNCTION prevent\_direct\_update\_melody\_signature\_str()

RETURNS TRIGGER AS $$

BEGIN

IF current\_setting('pg\_temp.trigger\_update\_melody', true) = 'true' THEN

RETURN NEW;

END IF;

*-- Nếu giá trị melody\_signature\_str bị thay đổi, chặn hành động*

IF NEW.melody\_signature\_str IS DISTINCT FROM OLD.melody\_signature\_str THEN

RAISE EXCEPTION 'Không được phép chỉnh sửa trực tiếp cột melody\_signature\_str. Hãy chỉnh sửa cột melody\_signature.';

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER trg\_prevent\_direct\_update\_melody\_signature\_str

BEFORE UPDATE OF melody\_signature\_str

ON songs

FOR EACH ROW

EXECUTE FUNCTION prevent\_direct\_update\_melody\_signature\_str();

* Cài đặt tự động cập nhật melody\_signature\_str khi melody\_signature thay đổi hoặc được thêm:

CREATE OR REPLACE FUNCTION update\_melody\_signature\_str()

RETURNS TRIGGER AS $$

BEGIN

*-- Chỉ cập nhật melody\_signature\_str nếu melody\_signature thay đổi*

IF NEW.melody\_signature IS DISTINCT FROM OLD.melody\_signature THEN

PERFORM set\_config('pg\_temp.trigger\_update\_melody', 'true', true);

NEW.melody\_signature\_str = array\_to\_string(NEW.melody\_signature, ',');

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER trg\_update\_melody\_signature\_str

BEFORE INSERT OR UPDATE

ON songs

FOR EACH ROW

EXECUTE FUNCTION update\_melody\_signature\_str();

* Cài đặt ràng buộc (5):
* Cài đặt không cho trực tiếp sửa song\_count, khi tạo danh sách phát mới, song\_count tự động bằng 0:

CREATE OR REPLACE FUNCTION prevent\_direct\_update\_song\_count()

RETURNS TRIGGER AS $$

BEGIN

*-- Bỏ qua nếu thao tác được thực hiện bởi trigger (được đánh dấu qua biến session pg\_temp)*

IF current\_setting('pg\_temp.trigger\_update', true) = 'true' THEN

RETURN NEW;

END IF;

*-- Khi thêm mới (INSERT)*

IF TG\_OP = 'INSERT' THEN

*-- Nếu giá trị song\_count được cung cấp và khác 0 -> Báo lỗi*

IF NEW.song\_count IS NOT NULL AND NEW.song\_count <> 0 THEN

RAISE EXCEPTION 'Không được phép thêm trực tiếp số lượng bài nhạc khác 0. Giá trị mặc định là 0.';

END IF;

*-- Đảm bảo song\_count luôn được khởi tạo bằng 0*

NEW.song\_count = 0;

*-- Khi cập nhật (UPDATE)*

ELSIF TG\_OP = 'UPDATE' THEN

*-- Nếu giá trị song\_count bị thay đổi -> Báo lỗi*

IF NEW.song\_count <> OLD.song\_count THEN

RAISE EXCEPTION 'Không được phép chỉnh sửa trực tiếp số lượng bài nhạc.';

END IF;

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER trg\_prevent\_direct\_update\_song\_count

BEFORE INSERT OR UPDATE

ON playlists

FOR EACH ROW

EXECUTE FUNCTION prevent\_direct\_update\_song\_count();

* Cài đặt để mỗi khi thêm bài nhạc vào danh sách phát, song\_count tự động tăng thêm 1

CREATE OR REPLACE FUNCTION increment\_song\_count()

RETURNS TRIGGER AS $$

BEGIN

*-- Đánh dấu rằng thao tác cập nhật được thực hiện bởi trigger*

PERFORM set\_config('pg\_temp.trigger\_update', 'true', true);

*-- Tăng số lượng bài nhạc trong playlists*

UPDATE playlists

SET song\_count = song\_count + 1

WHERE playlist\_id = NEW.playlist\_id;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER trg\_increment\_song\_count

BEFORE INSERT OR UPDATE ON playlist\_details

FOR EACH ROW

EXECUTE FUNCTION increment\_song\_count();

* Cài đặt để khi xóa bài nhạc khỏi danh sách phát, song\_count tự động giảm 1

CREATE OR REPLACE FUNCTION update\_song\_count\_on\_delete()

RETURNS TRIGGER AS $$

BEGIN

PERFORM set\_config('pg\_temp.trigger\_update', 'true', true);

*-- Giảm giá trị song\_count trong bảng playlists tương ứng*

UPDATE playlists

SET song\_count = song\_count - 1

WHERE playlist\_id = OLD.playlist\_id;

RETURN OLD;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER trigger\_update\_song\_count\_on\_delete

AFTER DELETE ON playlist\_details

FOR EACH ROW

EXECUTE FUNCTION update\_song\_count\_on\_delete();

* Cài đặt ràng buộc (6):
* Cài đặt không cho sửa trực tiếp position của bài nhạc:

CREATE OR REPLACE FUNCTION prevent\_update\_position()

RETURNS TRIGGER AS $$

BEGIN

*-- Bỏ qua nếu biến pg\_temp.ignore\_position\_update được thiết lập*

IF current\_setting('pg\_temp.ignore\_position\_update', true) = 'true' THEN

RETURN NEW;

END IF;

*-- Kiểm tra nếu cột position bị thay đổi*

IF NEW.position <> OLD.position THEN

RAISE EXCEPTION 'Không được phép chỉnh sửa trực tiếp cột position trong bảng playlist\_details.';

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER trg\_prevent\_update\_position

BEFORE UPDATE OF position

ON playlist\_details

FOR EACH ROW

EXECUTE FUNCTION prevent\_update\_position();

* Cài đặt mỗi khi thêm bài nhạc vào danh sách phát, thứ tự của bài nhạc đó ở cuối danh sách phát

CREATE OR REPLACE FUNCTION set\_position\_in\_playlist\_details()

RETURNS TRIGGER AS $$

DECLARE

current\_song\_count INTEGER;

BEGIN

*-- Lấy giá trị song\_count từ bảng playlists với playlist\_id tương ứng*

SELECT song\_count

INTO current\_song\_count

FROM playlists

WHERE playlist\_id = NEW.playlist\_id;

*-- Gán giá trị position là song\_count + 1*

NEW.position := current\_song\_count + 1;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER trg\_set\_position\_in\_playlist\_details

BEFORE INSERT

ON playlist\_details

FOR EACH ROW

EXECUTE FUNCTION set\_position\_in\_playlist\_details();

* Cài đặt để khi xóa một bài nhạc khỏi danh sách phát, thứ tự các bài nhạc còn lại trong danh sách phát vẫn đúng logic

CREATE OR REPLACE FUNCTION update\_position\_on\_delete()

RETURNS TRIGGER AS $$

BEGIN

PERFORM set\_config('pg\_temp.ignore\_position\_update', 'true', true);

*-- Giảm giá trị position của các hàng có cùng playlist\_id và position lớn hơn*

UPDATE playlist\_details

SET position = position - 1

WHERE playlist\_id = OLD.playlist\_id

AND position > OLD.position;

RETURN OLD;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER trigger\_update\_position\_on\_delete

AFTER DELETE

ON playlist\_details

FOR EACH ROW

EXECUTE FUNCTION update\_position\_on\_delete();

* Cài đặt procedure để đổi thứ tự của hai bài nhạc, không cho đổi trực tiếp mà chỉ có thể dùng procedure này để đổi

CREATE OR REPLACE PROCEDURE swap\_song\_positions(playlist\_id\_input INTEGER, song\_id\_1 INTEGER, song\_id\_2 INTEGER)

LANGUAGE plpgsql AS $$

DECLARE

position\_1 INTEGER;

position\_2 INTEGER;

BEGIN

*-- Thiết lập biến pg\_temp.ignore\_position\_update để cho phép cập nhật*

PERFORM set\_config('pg\_temp.ignore\_position\_update', 'true', true);

*-- Lấy giá trị position của song\_id\_1*

SELECT position INTO position\_1

FROM playlist\_details

WHERE playlist\_id = playlist\_id\_input AND song\_id = song\_id\_1;

*-- Lấy giá trị position của song\_id\_2*

SELECT position INTO position\_2

FROM playlist\_details

WHERE playlist\_id = playlist\_id\_input AND song\_id = song\_id\_2;

*-- Kiểm tra nếu một trong hai bài nhạc không tồn tại*

IF position\_1 IS NULL OR position\_2 IS NULL THEN

RAISE EXCEPTION 'Một hoặc cả hai bài nhạc không tồn tại trong playlist\_id = %', playlist\_id\_input;

END IF;

*-- Hoán đổi giá trị position*

UPDATE playlist\_details

SET position = position\_2

WHERE playlist\_id = playlist\_id\_input AND song\_id = song\_id\_1;

UPDATE playlist\_details

SET position = position\_1

WHERE playlist\_id = playlist\_id\_input AND song\_id = song\_id\_2;

*-- Hoàn thành hoán đổi*

RAISE NOTICE 'Đã hoán đổi vị trí của song\_id\_1 = % và song\_id\_2 = % trong playlist\_id = %',

song\_id\_1, song\_id\_2, playlist\_id\_input;

END;

$$;

* Cài đặt ràng buộc (7)
* Không được sửa playlist\_id

CREATE OR REPLACE FUNCTION prevent\_update\_playlist\_id()

RETURNS TRIGGER AS $$

BEGIN

RAISE EXCEPTION 'Không được phép thay đổi playlist\_id trong bảng playlist\_details.';

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER trg\_prevent\_update\_playlist\_id

BEFORE UPDATE OF playlist\_id

ON playlist\_details

FOR EACH ROW

EXECUTE FUNCTION prevent\_update\_playlist\_id();

* Không được sửa song\_id

CREATE OR REPLACE FUNCTION prevent\_update\_song\_id()

RETURNS TRIGGER AS $$

BEGIN

RAISE EXCEPTION 'Không được phép thay đổi song\_id trong bảng playlist\_details.';

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER trg\_prevent\_update\_song\_id

BEFORE UPDATE OF song

ON playlist\_details

FOR EACH ROW

EXECUTE FUNCTION prevent\_update\_song\_id();

# **CHƯƠNG 6: FUNCTION, STORED PROCEDURE, VIEW**

1. **Function**
2. Xác minh đăng nhập

CREATE OR REPLACE FUNCTION verify\_login(

email\_input TEXT,

password\_hash\_input TEXT

)

RETURNS TABLE(user\_id INTEGER, username CHARACTER VARYING(100)) AS $$

BEGIN

*-- Truy vấn để xác minh thông tin đăng nhập*

RETURN QUERY

SELECT u.user\_id, u.username

FROM users u

WHERE u.email = email\_input AND u.password\_hash = password\_hash\_input;

END;

$$ LANGUAGE plpgsql;

1. Tìm kiếm bài nhạc qua giai điệu

CREATE OR REPLACE FUNCTION search\_by\_melody(melody\_signature\_str\_input TEXT)

RETURNS TABLE(song\_title TEXT, song\_audio\_url TEXT) AS $$

BEGIN

RETURN QUERY

SELECT s.title::TEXT AS song\_title, s.audio\_url::TEXT AS song\_audio\_url

FROM songs s

WHERE s.melody\_signature\_str LIKE '%,' || melody\_signature\_str\_input || ',%'

OR s.melody\_signature\_str LIKE melody\_signature\_str\_input || ',%'

OR s.melody\_signature\_str LIKE '%,' || melody\_signature\_str\_input

OR s.melody\_signature\_str = melody\_signature\_str\_input;

END;

$$ LANGUAGE plpgsql;

1. Truy vấn lịch sử nghe nhạc của người dùng

CREATE OR REPLACE FUNCTION get\_listening\_history(user\_id\_input INTEGER)

RETURNS TABLE(

user\_id INTEGER,

song\_id INTEGER,

listened\_at TIMESTAMP

) AS $$

BEGIN

*-- Truy vấn dữ liệu từ bảng listening\_history*

RETURN QUERY

SELECT l.user\_id, l.song\_id, l.listened\_at

FROM listening\_history l

WHERE l.user\_id = user\_id\_input;

END;

$$ LANGUAGE plpgsql;

1. **Stored procedure**
2. Đổi thứ tự bài nhạc trong danh sách phát

Đã làm ở phần ràng buộc (6)

1. Xóa danh sách phát

Phải xóa tất cả bài nhạc thuộc danh sách phát đó rồi mới xóa danh sách phát (xóa playlist\_detail rồi mới xóa playlist)

CREATE OR REPLACE PROCEDURE delete\_playlist(playlist\_id\_input INTEGER)

LANGUAGE plpgsql AS $$

BEGIN

*-- Xóa tất cả các mục trong playlist\_details có playlist\_id*

DELETE FROM playlist\_details

WHERE playlist\_id = playlist\_id\_input;

*-- Xóa playlist trong bảng playlists*

DELETE FROM playlists

WHERE playlist\_id = playlist\_id\_input;

*-- Thông báo hoàn thành*

RAISE NOTICE 'Playlist với playlist\_id = % đã được xóa thành công cùng với các chi tiết liên quan.', playlist\_id\_input;

END;

$$;

1. Đăng ký tài khoản

CREATE OR REPLACE PROCEDURE register\_user(

email\_input CHARACTER VARYING(255),

username\_input CHARACTER VARYING(100),

password\_hash\_input CHARACTER VARYING(255),

OUT result INTEGER

)

LANGUAGE plpgsql AS $$

BEGIN

*-- Thêm thông tin người dùng vào bảng users*

BEGIN

INSERT INTO users (email, username, password\_hash)

VALUES (email\_input, username\_input, password\_hash\_input);

*-- Nếu thêm thành công, trả về 1*

result := 1;

EXCEPTION

WHEN unique\_violation THEN

*-- Nếu vi phạm UNIQUE constraint, in thông báo và trả về 0*

RAISE NOTICE 'Đăng ký thất bại: Email "%s" đã tồn tại.', email\_input;

result := 0;

END;

END;

$$;

1. **VIEW**
2. Top 10 bài nhạc được phát nhiều nhất

SELECT s.song\_id,

s.title,

count(l.song\_id) AS listen\_count

FROM songs s,

listening\_history l

WHERE l.song\_id = s.song\_id

GROUP BY s.song\_id, s.title

ORDER BY (count(l.song\_id)) DESC

LIMIT 10;

1. Các danh sách phát của một người dùng

CREATE OR REPLACE VIEW playlists\_of AS

SELECT p.playlist\_id,

p.user\_id,

p.name,

p.song\_count

FROM playlists p

* Ví dụ cách dùng:

SELECT \* from playlists\_of p WHERE p.user\_id = 105004

1. Top 10 tác giả được nghe nhiều nhất

CREATE OR REPLACE VIEW top\_10\_most\_listened\_authors AS

SELECT

a.author\_id,

a.name AS author\_name,

COUNT(lh.song\_id) AS total\_listens

FROM

authors a

JOIN

compositions c ON a.author\_id = c.author\_id

JOIN

songs s ON c.song\_id = s.song\_id

JOIN

listening\_history lh ON s.song\_id = lh.song\_id

GROUP BY

a.author\_id, a.name

ORDER BY

total\_listens DESC

LIMIT 10;

1. Các bài nhạc trong một danh sách phát

CREATE OR REPLACE VIEW songs\_of\_a\_playlist AS

SELECT pd.song\_id,

pd.playlist\_id,

s.title,

pd.position

FROM playlist\_details pd, songs s

WHERE pd.song\_id = s.song\_id

* Ví dụ cách dùng:

SELECT \* FROM songs\_of\_a\_playlist WHERE playlist\_id = 525433

1. Các bài nhạc của một tác giả

CREATE OR REPLACE VIEW songs\_of\_an\_author AS

SELECT s.song\_id,

a.author\_id,

s.title

FROM songs s, compositions c, authors a

WHERE s.song\_id = c.song\_id AND c.author\_id = a.author\_id

* Ví dụ cách dùng:

SELECT \* FROM songs\_of\_an\_author WHERE author\_id = 22502

# **CHƯƠNG 7: PHÂN QUYỀN CƠ SỞ DỮ LIỆU**

1. **Phân tích**

* Cơ sở dữ liệu không có nhiều nhóm đối tượng người dùng khác nhau, mỗi người dùng đều có quyền trên các bảng giống nhau
* Kết luận: tạo 2 role là admin\_user và app\_user

admin\_user: Quản trị viên, có toàn quyền trên cơ sở dữ liệu

app\_user: Tài khoản để những người dùng ứng dụng kết nối tới, có quyền xem, thêm, sửa, xóa ở các bảng users, playlists, playlist\_details, có quyền xem và thêm ở bảng listening\_history, và chỉ được quyền xem ở bảng songs, compositions và authors

1. **Code cài đặt**

CREATE ROLE admin\_user WITH LOGIN PASSWORD 'adminpassword';

GRANT ALL PRIVILEGES ON DATABASE "SearchByMelody" TO admin\_user;

CREATE ROLE app\_user WITH LOGIN PASSWORD 'userpassword';

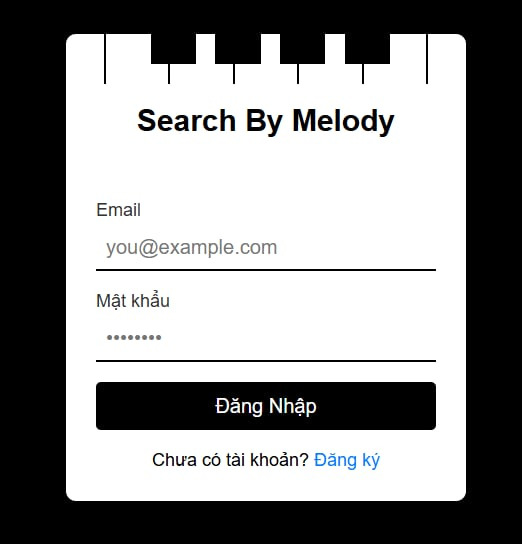
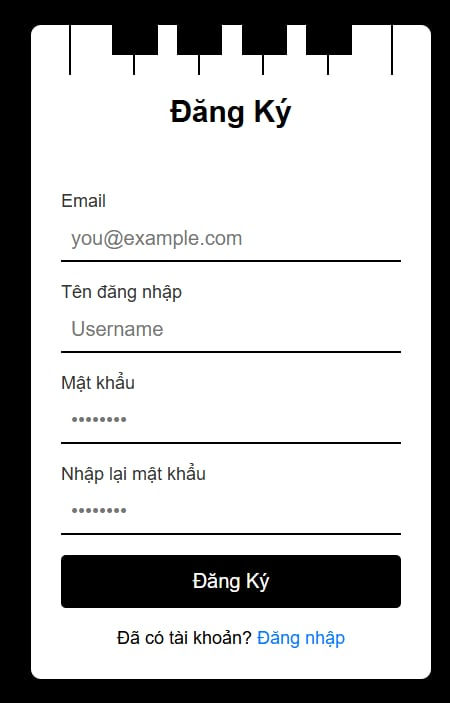
GRANT SELECT, INSERT, UPDATE, DELETE ON TABLE users, playlists, playlist\_details TO app\_user;

GRANT SELECT, INSERT ON TABLE listening\_history TO app\_user;

GRANT SELECT ON TABLE songs, compositions, authors TO app\_user;

# **CHƯƠNG 8: BIỂU DIỄN THÔNG TIN**

1. **Form đăng nhập, đăng ký tài khoản**

** **

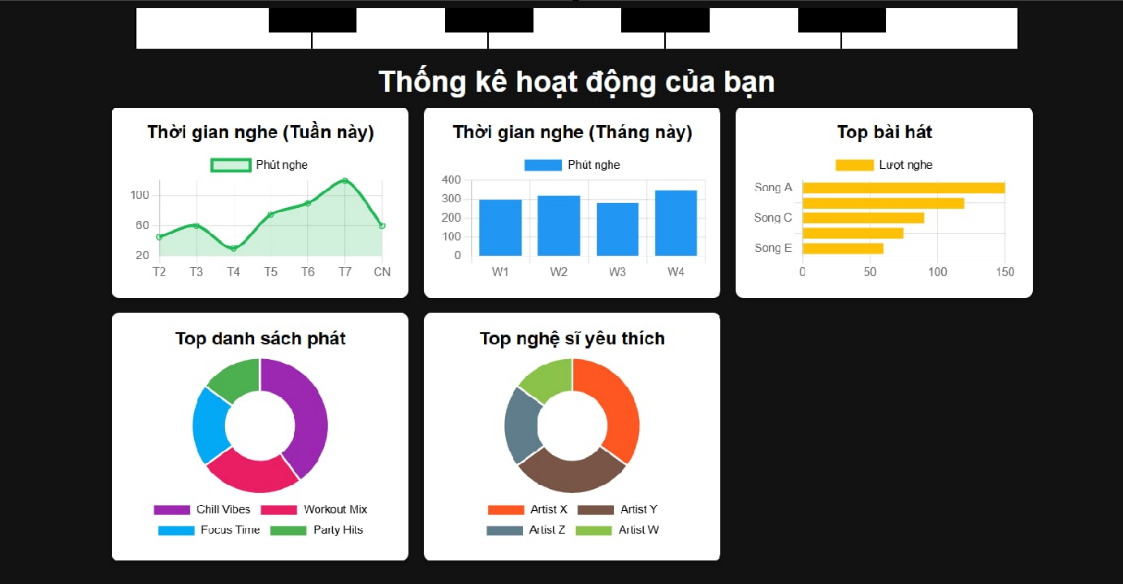
Hình 8.1: Form đăng nhập

1. **Menu các chức năng của ứng dụng**

****

Form 8.2: Menu chức năng

1. **Report thống kê hoạt động của người dùng**



Hình 8.3: Report thống kê hoạt động

# **CHƯƠNG 9: SAO LƯU, KHÔI PHỤC**

* Dùng cách thức đơn giản, kết nối với cơ sở dữ liệu qua pgadmin4 và export file backup
* Import file backup để khôi phục dữ liệu thông qua pgadmin4

1. **Sao lưu**

**A screenshot of a computer

AI-generated content may be incorrect.**

Hình 9.1: Giao diện sao lưu

1. **Khôi phục**

A screenshot of a computer

AI-generated content may be incorrect.

Hình 9.2: Giao diện khôi phục

# **CHƯƠNG 10: CÁC VẤN ĐỀ KHÁC**

1. **Đề xuất một cách thu thập dữ liệu cho cột đặc trưng giai điệu**

* Yêu cầu thu thập dữ liệu: trích xuất được ra khoảng cách cao độ (tính theo cung) giữa các nốt nhạc liền nhau trong bài nhạc. Lưu ý, nốt nhạc phải là giai điệu chính, không có nốt nhạc của nhạc đệm, bè
* Lựa chọn nguồn để trích xuất dữ liệu: các file MusicXML vì chúng là định dạng file XML, dễ dàng truy xuất thông tin, chúng cũng lưu giai điệu chính tách bạch với nhạc đệm
* Giải pháp: viết tool bằng python để tự động trích xuất mảng đặc trưng giai điệu

Code cài đặt như sau:

import os

import json

from music21 import converter, note, chord, stream

def find\_melody\_part(score):

    """

    Tìm phần (part) chứa giai điệu chính trong file MusicXML.

    """

    melody\_part = None

    max\_notes = 0

    for part in score.parts:

        # Kiểm tra nhãn của part (nếu có)

        if part.partName and "melody" in part.partName.lower():

            return part

        # Đếm số lượng nốt nhạc trong part

        num\_notes = len([n for n in part.flat.notes if isinstance(n, note.Note)])

        if num\_notes > max\_notes:

            max\_notes = num\_notes

            melody\_part = part

    return melody\_part

def extract\_intervals\_from\_melody(file\_path):

    """

    Trích xuất khoảng cách độ cao giữa các nốt liền nhau trong giai điệu chính từ file MusicXML.

    """

    try:

        # Đọc file MusicXML

        score = converter.parse(file\_path)

        # Tìm phần giai điệu chính

        melody\_part = find\_melody\_part(score)

        if melody\_part is None:

            print(f"Không tìm thấy giai điệu trong file {file\_path}.")

            return None

        # Lọc các nốt nhạc (bỏ qua hợp âm, nhịp nghỉ)

        melody\_notes = []

        for element in melody\_part.flat.notes:

            if isinstance(element, note.Note):

                melody\_notes.append(element)

            elif isinstance(element, chord.Chord):

                # Lấy nốt có cao độ lớn nhất trong hợp âm

                highest\_note = max(element.notes, key=lambda n: n.pitch.midi)

                melody\_notes.append(highest\_note)

        # Tính khoảng cách độ cao giữa các nốt liền nhau (theo cung)

        intervals = []

        for i in range(len(melody\_notes) - 1):

            pitch1 = melody\_notes[i].pitch.midi  # Cao độ nốt hiện tại (MIDI number)

            pitch2 = melody\_notes[i + 1].pitch.midi  # Cao độ nốt tiếp theo (MIDI number)

            interval = (pitch2 - pitch1) / 2  # Chuyển đổi từ MIDI number sang cung

            intervals.append(interval)

        return intervals

    except Exception as e:

        print(f"Lỗi khi xử lý file {file\_path}: {e}")

        return None

def process\_musicxml\_folder(folder\_path):

    """

    Xử lý tất cả các file MusicXML (.mxl) trong thư mục và trích xuất đặc trưng.

    """

    all\_intervals = {}

    for file\_name in os.listdir(folder\_path):

        if file\_name.endswith('.mxl'):

            file\_path = os.path.join(folder\_path, file\_name)

            print(f"Đang xử lý file: {file\_name}")

            intervals = extract\_intervals\_from\_melody(file\_path)

            if intervals is not None:

                all\_intervals[file\_name] = intervals

    return all\_intervals

def save\_intervals\_to\_json(data, output\_file):

    """

    Lưu kết quả trích xuất vào file JSON.

    """

    with open(output\_file, 'w', encoding='utf-8') as f:

        json.dump(data, f, ensure\_ascii=False, indent=4)

def main():

    # Đường dẫn tới thư mục chứa các file MusicXML

    folder\_path = r"D:\MusicXMLFolder"

    # Kiểm tra thư mục có tồn tại không

    if not os.path.isdir(folder\_path):

        print("Thư mục không tồn tại. Vui lòng kiểm tra lại.")

        return

    # Đường dẫn tới file JSON để lưu kết quả

    output\_file = r"D:\MusicXMLFolder\output.json"

    # Xử lý các file MusicXML và lưu kết quả

    print("Bắt đầu xử lý các file .mxl...")

    intervals\_data = process\_musicxml\_folder(folder\_path)

    save\_intervals\_to\_json(intervals\_data, output\_file)

    print(f"Đã lưu kết quả trích xuất vào {output\_file}")

if \_\_name\_\_ == "\_\_main\_\_":

    main()

1. **Hướng phát triển**

* Cách trích đặc trưng giai điệu này có thể dùng để tạo bộ dữ liệu training model AI có khả năng sáng tác nhạc
* Thêm nhiều chức năng khác cho việc quản lý danh sách phát như chia sẻ danh sách phát, đồng sở hữu danh sách phát, tạo danh sách phát phù hợp dựa vào gu nghe nhạc của nhiều người,…
* Tối ưu hiệu suất của truy vấn khó là tìm kiếm bài nhạc qua giai điệu
* Tối ưu cách lưu đặc trưng giai điệu bằng cách rút gọn những phần trùng lặp trong giai điệu (ví dụ như nhiều câu hát cùng giai điệu, hoặc là điệp khúc lặp lại hoặc lời một, lời hai nhưng giai điệu giống nhau…)

**CHƯƠNG 11: KẾT LUẬN**

1. **Tự đánh giá ưu điểm**

Đồ án của nhóm đã giải quyết được bài toán đặt ra:

* Nghĩ ra giải pháp số hóa dữ liệu giai điệu âm nhạc và cách truy vấn trên dữ liệu đó bằng phương pháp đơn giản, dễ triển khai
* Xây dựng một cơ sở dữ liệu phục vụ được các chức năng cần có của ứng dụng, tạo các ràng buộc để đảm bảo tính toàn vẹn dữ liệu, phân quyền để kiểm soát hành vi trong cơ sở dữ liệu
* Tạo được bản web demo chức năng tìm kiếm bài nhạc qua giai điệu

1. **Tự đánh giá khuyết điểm**

Vẫn còn những thứ cần cải thiện:

* Chưa tối ưu được truy vấn khó: tìm kiếm bài nhạc thông qua giai điệu
* Chưa thu thập số lượng lớn dữ liệu thực tế được và phải fake dữ liệu vì chưa thu thập được nhiều file MusicXML