ĐẠI HỌC QUỐC GIA TP.HCM
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC VÀ KĨ THUẬT MÁY TÍNH

Đồ án tổng hợp - hướng Kỹ thuật Dữ liệu (CO3127)

Đề tài: LinkedIn Job Postings (2023 - 2024)

Nhóm 2 - L05 - HK241

Giảng viên hướng dẫn: Trần Thị Quế Nguyệt
Sinh viên: Phạm Gia Nguyên ***(Leader)*** - 2212319
Trịnh Đình Khải - 2211561
Trương Nguyễn Minh Nhiên - 2212452
Phạm Duy Hưng - 2211379
Nguyễn Kim Đại Nghĩa - 2212229

Thành phố Hồ Chí Minh, 10/2024

Danh sách thành viên và phần trăm đóng góp

STT	Sinh viên	MSSV	Phần trăm
1	Phạm Gia Nguyên	2212319	21%
2	Trịnh Đình Khải	2211561	21%
3	Trương Nguyễn Minh Nhiên	2212452	21%
4	Phạm Duy Hưng	2211379	21%
5	Nguyễn Kim Đại Nghĩa	2212229	16%

Contents

1 Tiến độ công việc qua các giai đoạn **4**

2 Phân tích nhu cầu lưu trữ dữ liệu và thiết kế ERD **5**
2.1 Mô tả thông tin của bộ dữ liệu . . . 5
2.2 Phân tích nhu cầu lưu trữ dữ liệu . . . 7
2.3 Thiết kế ERD . . . 9
2.4 Các thực thể, thuộc tính và các nghiệp vụ của hệ thống . . . 9
2.5 Các ràng buộc dữ liệu của hệ thống . . . 10
2.6 Các ràng buộc ngữ nghĩa và nghiệp vụ của hệ thống . . . 11
2.7 Các thuộc tính dẫn xuất của hệ thống . . . 12

3 Hiện thực vào hệ quản trị DBMS **13**
3.1 Xác định các ràng buộc khoá chính và khoá ngoại . . . 13
3.2 Transaction . . . 13
3.3 Bảo mật và sao lưu . . . 14
3.4 Trực quan hóa dữ liệu . . . 16
3.5 Index và tối ưu hóa truy vấn . . . 16
3.5.1 Index . . . 16
3.5.2 Tối ưu hóa truy vấn . . . 18
3.6 Cấu trúc lưu trữ . . . 18
3.6.1 Heap Storage (Lưu trữ kiểu đống) . . . 18
3.6.2 Sequential Storage (Lưu trữ tuần tự) . . . 19
3.6.3 Index Storage (Lưu trữ chỉ mục) . . . 19
3.6.4 Columnar Storage (Lưu trữ theo cột) . . . 19
3.6.5 Row-Oriented Storage (Lưu trữ theo hàng) . . . 19
3.6.6 Object Storage (Lưu trữ đối tượng) . . . 20
3.6.7 File-based Storage (Lưu trữ dựa trên tệp) . . . 20
3.6.8 In-Memory Storage (Lưu trữ trong bộ nhớ) . . . 20
3.6.9 Distributed Storage (Lưu trữ phân tán) . . . 20
3.7 Các check constraints . . . 21
3.8 Các trigger liên quan . . . 22

4 Xử lý các dữ liệu mẫu để import vào CSDL trên DBMS **26**
4.1 Các câu lệnh tạo bảng . . . 26
4.2 Các ràng buộc khoá ngoại trên các bảng . . . 30
4.3 Xử lý dữ liệu mẫu và import vào CSDL trên DBMS . . . 31

5 Xây dựng trang Content Management System **48**

6 Xây dựng giao diện Dashboard hiển thị các biểu đồ **53**

7 Ứng dụng kết nối cơ sở dữ liệu **57**
7.1 Tổng quan công nghệ . . . 57
7.1.1 Công nghệ sử dụng . . . 57
7.1.2 Tổ chức thư mục code . . . 57
7.1.2.a Tổ chức thư mục FrontEnd . . . 57
7.1.2.b Tổ chức thư mục BackEnd . . . 58
7.1.2.c Luồng xử lý phía BackEnd . . . 59

7.1.3 Khai báo kết nối với DBMS 59
7.2 Hiện thực 59
7.2.1 Job 59
7.2.2 Posting 64
7.2.3 Company 70
7.2.4 Dashboard 81

8 Kết luận **85**

1 Tiến độ công việc qua các giai đoạn

Task	Thời gian	Nhiệm vụ	Phân công
a.	22/09/2024	*2. Thảo luận đề tài và nghiên cứu dataset.*	**Cả nhóm**
	03/10/2024	2.1. Mô tả thông tin của bộ dữ liệu.	Nhiên
		2.2. Thảo luận phân tích nhu cầu lưu trữ dữ liệu.	**Cả nhóm**
		2.2. Phân tích dữ liệu phần Posting.	Nguyên, Hưng
		2.2. Phân tích dữ liệu phần Company.	Nhiên
		2.2. Phân tích dữ liệu phần Job.	Khải
	08/10/2024	2.3. Thiết kế ERD.	**Cả nhóm**
	15/10/2024	2.4. Các thực thể, thuộc tính, nghiệp vụ của hệ thống.	Khải
		2.5. Các ràng buộc dữ liệu của hệ thống.	Nhiên
		2.6. Các ràng buộc ngữ nghĩa, nghiệp vụ của hệ thống.	Nghĩa
		2.7. Các thuộc tính dẫn xuất của hệ thống.	Hưng
b.	20/10/2024	*3. Thảo luận hiện thực vào hệ quản trị DBMS.*	**Cả nhóm**
	27/10/2024	3.1. Xác định các ràng buộc khóa chính, khóa ngoại.	Khải, Hưng
		3.2. Đánh chỉ mục.	Nguyên, Nhiên
		3.3. Các check constraint và 3.4. Các trigger.	Nghĩa
	30/10/2024	*Thảo luận để thực hiện các task 4, task 5 và task 6.*	**Cả nhóm**
c.	03/11/2024	4. Xử lý lại dữ liệu mẫu để import vào DBMS.	**Cả nhóm**
d.	03/11/2024	5. Xây dựng trang Content Management System demo đọc dữ liệu, lưu trữ, xoá, cập nhật cho admin.	**Cả nhóm**
e.	03/11/2024	6. Xây dựng giao diện Dashboard hiển thị các biểu đồ.	**Cả nhóm**

Phân công cụ thể Task 5 và Task 6			
STT	**Thành viên**	**FrontEnd**	**BackEnd**
1	Nguyên	Dashboard 3. Posting Title.	Job, Dashboard 1+3.
2	Khải	Dashboard 5. Posting Count.	Company, Dashboard 5.
3	Nhiên	Dashboard 4. Job Title.	Posting, Dashboard 2+4.
4	Hưng	Dashboard 1+2. Worktype and Salary Distribution; Home, Navigation, Company.	Không
5	Nghĩa	Job, Posting.	Không

2 Phân tích nhu cầu lưu trữ dữ liệu và thiết kế ERD

2.1 Mô tả thông tin của bộ dữ liệu

- Tên bộ dữ liệu: LinkedIn Job Postings (2023 - 2024).
- Nguồn dữ liệu: Kaggle, link: https://bit.ly/LinkedIn-Job-Postings-2023-2024
- Mô tả bộ dữ liệu: với hơn 124.000 tin tuyển dụng từ 2023-2024 chứa các thông tin quan trọng về công việc, công ty và xu hướng tuyển dụng theo thời gian.
- Thông tin của các datatset:

1. **job_postings.csv** là file về các bài đăng tuyển dụng, gồm các thuộc tính như:
   - `job_id`: Mã công việc được LinkedIn định nghĩa.
   - `company_id`: Mã định danh cho công ty liên quan đến tin tuyển dụng.
   - `title`: Tiêu đề công việc.
   - `description`: Mô tả công việc.
   - `max_salary`: Mức lương tối đa.
   - `med_salary`: Mức lương trung bình.
   - `min_salary`: Mức lương tối thiểu.
   - `pay_period`: Kỳ hạn trả lương (theo giờ, theo tháng, theo năm).
   - `formatted_work_type`: Loại hình công việc (Toàn thời gian, Bán thời gian, Hợp đồng).
   - `location`: Địa điểm làm việc.
   - `applies`: Số lượng ứng viên đã nộp đơn.
   - `original_listed_time`: Thời gian ban đầu công việc được đăng.
   - `remote_allowed`: Công việc có cho phép làm việc từ xa không.
   - `views`: Số lần tin tuyển dụng đã được xem.
   - `job_posting_url`: URL dẫn đến tin tuyển dụng trên một nền tảng.
   - `application_url`: URL nơi ứng viên có thể nộp đơn.
   - `application_type`: Loại quy trình nộp đơn (ngoài trang, đơn giản/phức tạp trên trang).
   - `expiry`: Ngày hoặc thời gian tin tuyển dụng hết hạn.
   - `closed_time`: Thời gian tin tuyển dụng đóng.
   - `formatted_experience_level`: Trình độ kinh nghiệm (mới vào, cộng sự, quản lý, v.v.).
   - `skills_desc`: Mô tả chi tiết các kỹ năng yêu cầu cho công việc.
   - `listed_time`: Thời gian công việc được đăng.
   - `posting_domain`: Tên miền của trang web có đơn ứng tuyển.
   - `sponsored`: Tin tuyển dụng có được tài trợ hoặc quảng cáo không.
   - `work_type`: Loại công việc liên quan đến công việc.
   - `currency`: Loại tiền tệ mà mức lương được cung cấp.
   - `compensation_type`: Loại thù lao cho công việc.

2. **jobs/benefits.csv** là file gồm thông tin về phúc lợi được cung cấp cho công việc.
   - `job_id`: Mã công việc.
   - `type`: Loại phúc lợi được cung cấp (401K, Bảo hiểm y tế, v.v.).
   - `inferred`: Phúc lợi được gắn thẻ hoặc được suy ra từ văn bản của LinkedIn.
3. **jobs/job_industries.csv** là file chứa thông tin về mối liên kết giữa công việc và ngành công nghiệp.
   - `job_id`: Mã công việc.
   - `industry_id`: Mã ngành công nghiệp.
4. **jobs/job_skills.csv** là file chứa thông tin về các kỹ năng yêu cầu cho từng công việc.
   - `job_id`: Mã công việc.
   - `skill_abr`: Từ viết tắt của kỹ năng.
5. **jobs/salaries.csv** là file chứa thông tin về mức lương cho từng công việc.
   - `salary_id`: Mã mức lương.
   - `job_id`: Mã công việc.
   - `max_salary`: Mức lương tối đa.
   - `med_salary`: Mức lương trung bình.
   - `min_salary`: Mức lương tối thiểu.
   - `pay_period`: Kỳ hạn trả lương (theo giờ, theo tháng, theo năm).
   - `currency`: Đơn vị tiền tệ của mức lương.
   - `compensation_type`: Loại thù lao cho công việc.
6. **companies/companies.csv** là file chứa thông tin chi tiết về các công ty.
   - `company_id`: Mã công ty được LinkedIn định nghĩa.
   - `name`: Tên công ty.
   - `description`: Mô tả công ty.
   - `company_size`: Quy mô công ty dựa trên số lượng nhân viên (có giá trị từ 1 đến 7).
   - `state`: Bang của trụ sở chính công ty.
   - `country`: Quốc gia của trụ sở chính công ty.
   - `city`: Thành phố của trụ sở chính công ty.
   - `zip_code`: Mã ZIP của trụ sở chính công ty.
   - `address`: Địa chỉ trụ sở chính công ty.
   - `url`: Liên kết đến trang LinkedIn của công ty.
7. **companies/employee_counts.csv** là file chứa thông tin về số lượng nhân viên và người theo dõi của từng công ty.
   - `company_id`: Mã công ty.
   - `employee_count`: Số lượng nhân viên của công ty.
   - `follower_count`: Số lượng người theo dõi công ty trên LinkedIn.
   - `time_recorded`: Thời gian thu thập dữ liệu theo định dạng Unix.

8. **companies/company_industries.csv** là file chứa thông tin về các ngành công nghiệp mà các công ty thuộc về.
   - `company_id`: Mã công ty được LinkedIn định nghĩa.
   - `industry_name`: Tên ngành công nghiệp.
9. **companies/company_specialities.csv** là file chứa thông tin về các chuyên ngành của từng công ty.
   - `company_id`: Mã công ty được LinkedIn định nghĩa.
   - `speciality`: Tên lĩnh vực chuyên môn của công ty.
10. **mappings/industries.csv** là file hứa thông tin về các ngành công nghiệp.
    - `industry_id`: Mã ngành công nghiệp.
    - `industry_name`: Tên ngành công nghiệp.
11. **mappings/skills.csv** là file chứa thông tin về các kỹ năng.
    - `skill_abr`: Từ viết tắt của kỹ năng.
    - `skill_name`: Tên đầy đủ của kỹ năng.

2.2 Phân tích nhu cầu lưu trữ dữ liệu

1. **postings.csv**: gồm 123849 bản ghi và 29 trường thuộc tính.
   - `title`: Một công ty có thể có nhiều hơn một vị trí hoặc chức danh công việc (ngăn cách bằng dấu "," hoặc "/").
   - `remote_allowed`: Giá trị 1 (được làm việc từ xa) hoặc NULL (không cho phép làm việc từ xa).
   - `original_listed_time, expiry, listed_time`: Dạng Unix timestamp với số giây đã trôi qua kể từ 00:00:00 UTC ngày 01/01/1970. Chuyển đổi sang dạng DATE bằng cách chia cho 86400 (số giây trong một ngày) và cộng với DATE(1970,1,1).
   - `application_url, formatted_experience_level, posting_domain, close time`: Thường chứa nhiều giá trị NULL, nên tách ra bảng riêng để tối ưu lưu trữ.
   - Tách bảng **posting_state.csv** bao gồm các thuộc tính từ **posting.csv** như `applies`, `views`, `expriy`, `listed_time`, `original listed_time` và thêm thuộc tính dẫn xuất `apply_rate`, `remaining_time`, `timezone_offset`. Giúp phân tích trạng thái bài đăng, tỉ lệ người xem đăng kí ứng tuyển và định dạng thời gian theo thời gian thực tại các quốc gia.
2. **jobs/benefits.csv**: gồm 67943 bản ghi và 3 trường thuộc tính.
   - `inferred`: Giá trị 0 (phúc lợi được ghi rõ ràng trong mô tả công việc - explicitly tagged) và Giá trị 1 (phúc lợi không được nhắc đến trực tiếp - inferred through text).
   - `type`: Dữ liệu về các loại bảo hiểm bị lặp lại nhiều lần trong dataset hiện tại. Cần chuẩn hóa dữ liệu bằng cách tách một bảng mới **benefits_type**, gồm thuộc tính `type` từ file **jobs/benefits.csv** va thay thể thuộc tính `type` bằng `benefit_id` để ánh xạ đến lại dữ liệu. Việc này giúp giảm thiểu trùng lặp dữ liệu, dễ quản lý và tối ưu hóa hiệu suất xử lý.
3. **jobs/job_industries.csv**: gồm 164808 bản ghi và 2 trường thuộc tính.
   - `industry_id`: Mã định danh của các ngành công nghiệp được ánh xạ từ file **mappings/industries.csv**.

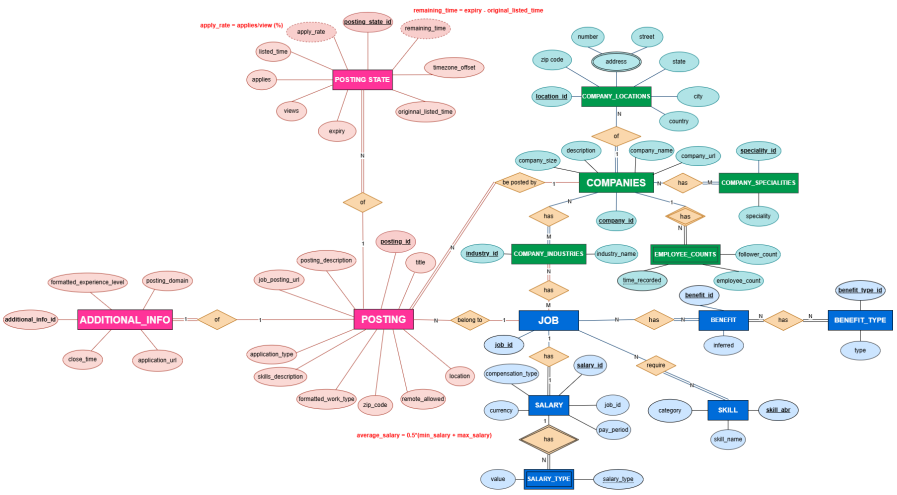
4. **jobs/jobs_skills.csv**: gồm 213768 bản ghi và 2 trường thuộc tính.
   - `skill_abr`: Gồm mã định danh (các ký tự viết tắt) được ánh xạ từ file **mappings/skills.csv**.
5. **jobs/salaries.csv**: gồm 40785 bản ghi và 8 trường thuộc tính.
   - `currency`: Mã tiền tệ theo chuẩn ISO-International Standard Organization (*ví dụ: USD, CAD, BBD, EUR, AUD, GBP,...*).
   - `max_salary`, `min_salary` và `med_salary`: Thường chứa nhiều giá trị NULL, cần tách bảng riêng **jobs/salaries_type** gồm các thuộc tính `salary_type` (gồm các giá trị max, min, med và average) và `salary_value` (giá trị tương ứng của từng loại lương). Giúp tối lưu trữ và quản lý dữ liệu về lương.
6. **companies/companies.csv** gồm 24473 bản ghi và 5 trường thuộc tính.
   - `name`: Có thể chứa ký tự đặc biệt như "&", "-", "|", cần xử lý để đảm bảo tính nhất quán. Chuẩn hóa tên công ty (*ví dụ: "B2BRevops.com", "Natcast.org"*) về đúng định dạng và loại bỏ quy tắc viết hoa, đảm bảo không chứa lỗi chính tả.
   - `description`: Thường dài và không đồng nhất, cần loại bỏ các ký tự không cần thiết (lỗi font, dấu xuống dòng, v.v) và chuẩn hóa định dạng để dễ xử lý.
   - `url`: Kiểm tra tính hợp lệ, đảm bảo tính chính xác và khả dụng.
7. **companies/company_industries.csv** gồm 24375 và 2 trường thuộc tính.
   - `industry_id`: Mã ngành công nghiệp từ file **mappings/industries**, dùng để giảm trùng lặp giữa các công ty.
   - `industry_name`: Thay bằng `industry_id` - mã ngành công nghiệp từ file **mappings/industries**, có thể trùng lặp giữa các công ty để giảm kích thước dữ liệu.
8. **companies/company_specialities.csv** gồm 169387 bản ghi và 2 trường thuộc tính.
   - `specialities`: cần chuẩn hóa để đảm bảo tính nhất quán. Do một số công ty có nhiều hơn một chuyên môn, được ngăn cách bởi các dấu ",", ";", "/" hoặc "&".
9. **companies/employee_counts.csv** gồm 35787 bản ghi và 4 trường thuộc tính.
   - `speciality_id`: Thêm mã định danh chuyên môn của công ty để dễ xác định thông tin và giảm kích thước dữ liệu.
   - `follower_count` và `employee_count`: Theo dõi thay đổi theo thời gian để đảm bảo dữ liệu cập nhật.
   - `time_recorded`: Chuyển đổi từ Unix timestamp sang định dạng dễ đọc, tương tự các thuộc tính thời gian trong **posting.csv**.
10. **companies/companies_location.csv** là file gồm những thuộc tính liên quan đến vị trí được tách ra từ **companies/companies** để nâng cao chất lượng dữ liệu và tính đồng nhất, gồm 24473 bản ghi và 6 trường thuộc tính.
    - `state`, `country`, `city`, `zip_code`, `number` và `street`: Có thể chứa giá trị "0" nếu thiếu thông tin.
    - `state`: Chuẩn hóa để thống nhất giữa tên đầy đủ, tên viết tắt và loại bỏ dữ liệu lỗi.
    - `country`: Chuẩn hóa theo mã viết tắt tiêu chuẩn ISO.
    - `city`: Loại bỏ thông tin không hợp lệ (`zip_code`, `state`, số, lỗi font) và chuẩn hóa định dạng. *Ví dụ: "CH-1009 PULLY/LAUSANNE", "San Diego - US | Tokyo - JP", "20+ Locations", v.v.*

- `zip_code`: Chuẩn hóa định dạng và loại bỏ thông tin sai trường. *Ví dụ: "TEL: 42000600", "New York, Boston, London, Tokyo", v.v.*
- `number`: Tách từ thuộc tính `address`, chỉ chứa ký tự số và có thể ngăn cách bởi dấu "-".
- `street`: Tách từ thuộc tính `address`, chứa ký tự chữ cái, có thể chứa ký tự đặc biệt ("+", ",", ":") và số tòa. Loại bỏ giá trị không hợp lệ như tên tiểu bang.

11. **mappings/industries.csv và mappings/skills.csv** là các file dùng để ánh xạ với các tệp file khác. Cụ thể, `industry_id` là mã ngành công nghiệp `industry_name`, tương tự `skill_abr` là từ viết tắt của kỹ năng `skill_name`.
    - `industry_name`: Chuẩn hóa để đảm bảo tính nhất quán, do một số lĩnh vực nhỏ hơn trong một ngành được ngăn cách với nhau bằng dấu ",". Một số còn có giá trị NULL.

2.3 Thiết kế ERD

Từ những phân tích yêu cầu hệ thống ở trên, nhóm thiết kế được sơ đồ **Entity Relationship Diagram (ERD)**. Link drive dẫn đến sơ đồ ERD trên draw.io: https://bit.ly/ERD-DATH-KTDL-Nhom02-L05.



Hình 1: Sơ đồ ERD của hệ thống cơ sở dữ liệu LinkedIn Job Postings.

2.4 Các thực thể, thuộc tính và các nghiệp vụ của hệ thống

Hệ thống cơ sở dữ liệu được thiết kế để tối ưu hóa việc quản lý và phân tích thông tin công ty, bài đăng tuyển dụng, công việc. Hệ thống cũng hỗ trợ theo dõi và đánh giá hiệu quả bài đăng, mức độ ứng tuyển, từ đó cải thiện khả năng truy vấn dữ liệu về lương, phúc lợi và các kỹ năng yêu cầu cho từng công việc.

1. **Quản lý công ty Company**

   Hệ thống cơ sở dữ liệu lưu trữ và quản lý thông tin về công ty, bao gồm các thuộc tính như mã công ty duy nhất, quy mô, tên công ty, mô tả, số lượng người theo dõi, URL LinkedIn và địa chỉ trụ sở công ty (**Company_Location**). Địa chỉ bao gồm quốc gia, thành phố, tiểu bang, mã bưu chính, số và tên đường.

Các công ty được liên kết với các chuyên môn (**Company_Specialities**) và ngành nghề (**Company_Industries**) qua các mã định danh duy nhất và tên riêng. Một công ty có thể có nhiều chuyên môn và nhiều ngành, đồng thời mỗi chuyên môn hoặc ngành nghề phải thuộc ít nhất một công ty. Hệ thống cũng cho phép theo dõi số lượng nhân viên và người theo dõi của công ty (**Employee_Counts**) tại các thời điểm cụ thể.

2. **Quản lý bài đăng tuyển dụng Posting**

   Hệ thống cho phép quản lý các bài đăng tuyển dụng với các thông tin như mã bài đăng, mã bưu chính, địa điểm, mô tả công việc, kỹ năng yêu cầu, tiêu đề, mô tả chi tiết bài đăng, URL và loại hình làm việc (có cho phép làm việc từ xa hay không). Hệ thống theo dõi trạng thái của mỗi bài đăng **(Posting_State)**, bao gồm lượt xem, lượt đăng ký, tỷ lệ đăng ký, ngày đăng, ngày hết hạn, thời gian cập nhật và thời gian còn hiệu lực. Thông tin thêm **(Additional_Info)** như mã bài đăng, kinh nghiệm yêu cầu, nền tảng đăng bài, URL ứng tuyển và ngày đóng nhận hồ sơ cũng được lưu trữ.

3. **Quản lý công việc Job**

   Hệ thống quản lý các công việc, với mỗi công việc có thể thuộc nhiều ngành và yêu cầu nhiều kỹ năng (**Skill**). Mỗi công việc có thông tin về bảng mức lương (**Salary**), bao gồm mã lương, loại bồi thường, kỳ thanh toán và loại tiền tệ. Lương được phân loại (**Salary_Type**) theo các mã và giá trị cụ thể. Công việc cũng có thể có nhiều phúc lợi khác nhau (**Benefit**), với mã phúc lợi duy nhất và thông tin xác định phúc lợi được ghi rõ hoặc suy ra từ mô tả bài đăng. Các loại phúc lợi (**Benefit_Type**) cũng được phân loại và lưu trữ với mã và giá trị tương ứng.

2.5 Các ràng buộc dữ liệu của hệ thống

Các ràng buộc dữ liệu của hệ thống nhằm đảm bảo tính nhất quán và chính xác trong hệ thống. Các ràng buộc này giúp duy trì tính toàn vẹn dữ liệu với các khóa và hạn chế lỗi của giá trị đầu vào, như các thuộc tính số học, chuỗi, và thời gian.

1. **Các ràng buộc về khóa được thể hiện trong ERD.**
2. **Các thuộc tính liên quan đến lương:** `max_salary`, `med_salary`, `min_salary` phải là số thực không âm và đảm bảo rằng `max_salary` $\geq$ `med_salary` $\geq$ `min_salary`.
3. **Các thuộc tính liên quan đến số lượng:** `employee_count`, `follower_count`, `views`, `applies`, `company_size` phải là số nguyên không âm.
4. **Các thuộc tính liên quan đến thời gian:** `listed_time`, `expiry`, `original_listed_time` phải tuân theo định dạng hợp lệ là `dd/mm/yyyy` và đảm bảo `expiry` $>$ `listed_time` $\geq$ `original_listed_time`.
5. **Các thuộc tính liên quan đến trạng thái:**
   - `remote_allowed` chỉ nhận hai giá trị "0" và "1".
   - `formatted_work_type` nhận các giá trị như "Full-time", "Part-time", "Internship", "Temporary", "Contract", "Volunteer", "Other".
   - `application_type` chỉ nhận ba giá trị "ComplexOnsiteApply", "OffsiteApply", "SimpleOnsiteApply".
   - `pay_period` chỉ nhận ba giá trị "YEARLY", "HOURLY" và "MONTHLY".

- `compensation_type` chỉ nhận giá trị "BASE_SALARY".
- `company_size` chỉ nhận giá trị từ 0 đến 7.
- `apply_rate` được tính theo đơn vị %.
- `salary_type` chỉ gồm 3 giá trị là max, min, med, average

6. **Các thuộc tính liên quan đến vị trí:** `zip_code` và `location` phải có định dạng phù hợp; `zip_code` phải là mã bưu chính hợp lệ, tương ứng với `location`; `state`, `country`, `city` phải tồn tại và đúng với vị trí địa lý thực tế.
7. **Thuộc tính `company_name`, `title`:** phải có độ dài hợp lý (giới hạn từ 3 đến 255 ký tự).
8. **Thuộc tính `job_posting_url`, `application_url`:** phải có định dạng hợp lý của một đường dẫn url.
9. **Thuộc tính `address`:** phải có định dạng hợp lệ, gồm hai phần: `number` phải là số nguyên không âm và `street` chỉ chứa ký tự chữ cái.
10. **Các giá trị mặc định:**
    - `applies`, `remote_allowed`: mặc định là 0.
    - Các trường về text thì mặc định là chuỗi rỗng.
    - Các trường còn lại mặc định là `Null`.

2.6 Các ràng buộc ngữ nghĩa và nghiệp vụ của hệ thống

Một số ràng buộc ngữ nghĩa không thể hiện được ERD hệ thống:

1. **Company** chỉ có thể yêu cầu **Skill** thuộc về các **Speciality** mà công ty chuyên về.
2. Mỗi **Posting** có thể yêu cầu các **Skill** từ `skill_description` và phải liên kết với các thuộc tính hợp lệ có trong **Skill**.
3. Địa điểm `location` và mã bưu chính `zip_code` trong bài đăng tuyển dụng **Posting** phải khớp với địa điểm **Location** và mã bưu chính `zip_code` của công ty **Company**.
4. Mức lương **Salary** dựa theo `formatted_experience_type`, nhân viên có kinh nghiệm phải cao hơn mức lương của nhân viên ít kinh nghiệm.
5. Mức lương **Salary** của nhân viên trong cùng một ngành nghề **Job** với hình thức làm việc `application_type` khác nhau: Full-time > Part-time > Internship > Volunteer.
6. Trong cùng một ngành nghề **Job** thì tiền lương trả theo: YEARLY > MONTHLY > HOURLY.
7. Trong cùng một ngành nghề **Job**, yêu cầu nhiều **Skill** hơn thì lương **Salary** cao hơn.
8. Một công ty **Company** không thể đăng nhiều bài đăng tuyển dụng **Posting** giống hệt nhau trong một thời gian ngắn.
9. Một công ty **Company** càng có nhiều ngành kinh doanh **Industry** thì quy mô công ty `company_size` càng lớn.
10. Một công ty **Company** có nhiều ngành kinh doanh **Industry** thì số lượng bài tuyển dụng phải nhiều.

11. **Salary** của **Job** thuộc **Company** phải phù hợp với mặt bằng của **Location** và phụ thuộc vào đặc thù của **Industry**.

12. **Company** có thể cung cấp nhiều loại **Benefit**, nhưng **Benefit** chỉ được áp dụng cho từng **Job** tại **Industry** mà **Company** vận hành.

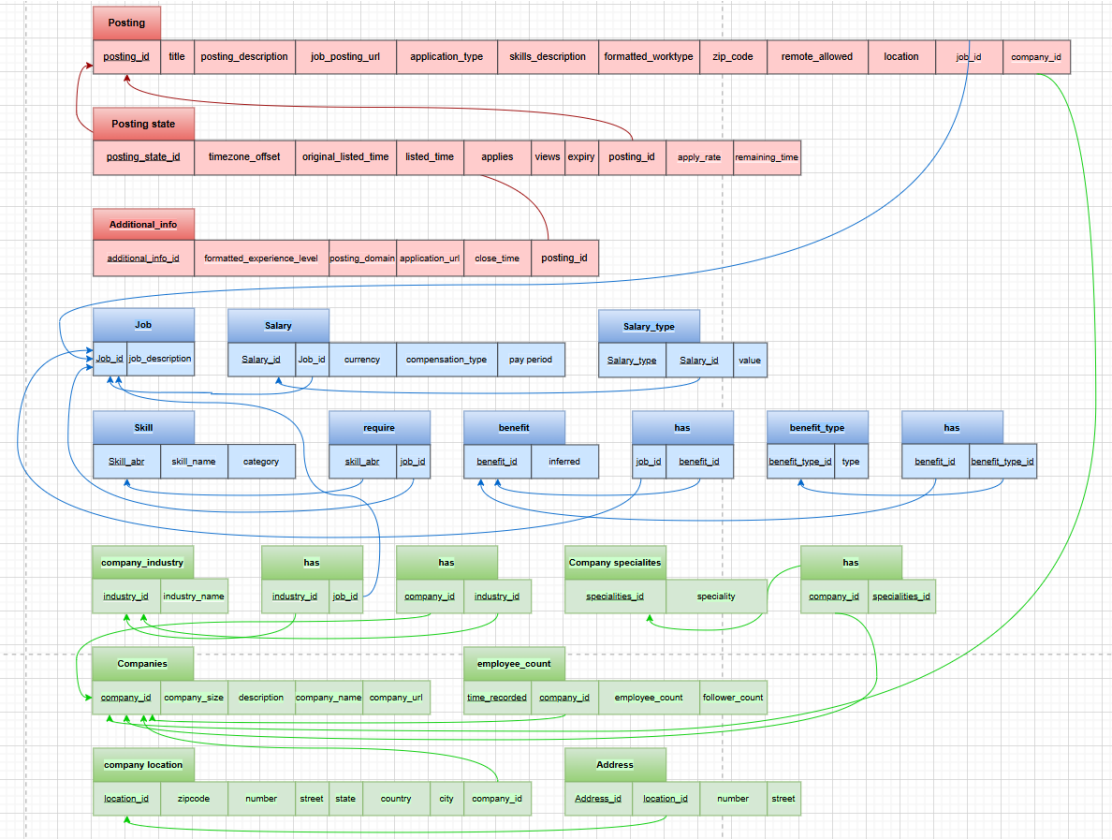
2.7 Các thuộc tính dẫn xuất của hệ thống

- **SALARY.**`average_salary` được tính bằng cách lấy trung bình của **SALARY.**`max_salary` và **SALARY.**`min_salary`.
- **POSTING.**`remaining_time` được tính toán bằng cách lấy giá trị của **TIME.**`expiry` trừ đi giá trị của **TIME.**`original_listed_time`.
- **POSTING.**`apply_rate` dùng **POSTING.**`applies` chia cho **POSTING.**`views`.

3 Hiện thực vào hệ quản trị DBMS

3.1 Xác định các ràng buộc khoá chính và khoá ngoại

Các ràng buộc về khóa chính và khóa ngoại được thể hiện rõ trong lược đồ ánh xạ dữ liệu. Link drive dẫn đến lược đồ ánh xạ trên draw.io: https://bit.ly/MAPPING-DATH-KTDLNhom02-L05.



Hình 2: Sơ đồ MAPPING của hệ thống cơ sở dữ liệu LinkedIn Job Postings.

3.2 Transaction

Trong cơ sở dữ liệu, **Transaction** là một chuỗi các thao tác thực hiện trên cơ sở dữ liệu, được thực hiện như một đơn vị công việc duy nhất.

1. **Tuân theo các tính chất ACID để đảm bảo tính toàn vẹn và nhất quán dữ liệu.**
   - ***Atomicity (Tính nguyên tử):*** Mỗi transaction là một đơn vị công việc không thể chia nhỏ. Tất cả các thao tác trong transaction được thực hiện thành công hoặc không có thao tác nào được thực hiện.
   - ***Consistency (Tính nhất quán):*** Transaction phải đưa cơ sở dữ liệu từ một trạng thái nhất quán này sang trạng thái nhất quán khác. Dữ liệu sau khi thực hiện transaction phải tuân theo tất cả các ràng buộc đã được định nghĩa trước.

- ***Isolation (Tính cô lập):*** Mỗi transaction phải thực hiện độc lập với các transaction khác. Transaction đang thực hiện không được phép nhìn thấy sự thay đổi chưa hoàn tất của các transaction khác.
- ***Durability (Tính bền vững):*** Khi một transaction đã được xác nhận (committed), những thay đổi của nó phải được lưu trữ vĩnh viễn, ngay cả khi hệ thống gặp sự cố.

2. **Các Lệnh kiểm soát transaction**
   - BEGIN TRANSACTION
   - COMMIT TRANSACTION
   - COMMIT WORK
   - ROLLBACK TRANSACTION
   - ROLLBACK WORK
   - SAVE TRANSACTION

3. **Thực hiện transaction:** SQL Server hỗ trợ các giao tác trong một số chế độ.
   - ***Tự động thực hiện giao tác:*** Mỗi câu lệnh đơn được thực hiện và tự động commit ngay sau khi nó hoàn thành, không cần viết bất kỳ câu lệnh cụ thể nào để bắt đầu và kết thúc các transaction. Mỗi câu lệnh được xem là 1 transaction, đây là chế độ mặc định cho SQL Server Database Engine.
   - ***Các transaction rõ ràng:*** Mọi transaction bắt đầu rõ ràng với câu lệnh **BEGIN TRANSACTION** và kết thúc với giao tác **ROLLBACK** hoặc **COMMIT**.
   - ***Các transaction ẩn:*** Một transaction mới sẽ tự động bắt đầu khi transaction trước đó hoàn thành và được hoàn thành một cách rõ ràng bằng cách sử dụng câu lệnh **ROLLBACK** hoặc **COMMIT**.

3.3 Bảo mật và sao lưu

Trong cơ sở dữ liệu, **Database Security (Bảo mật Dữ liệu)** là một khía cạnh quan trọng để bảo vệ dữ liệu khỏi các truy cập trái phép, tấn công hoặc mất mát. Một số phương pháp và khái niệm phổ biến để đảm bảo bảo mật cho hệ thống cơ sở dữ liệu:

1. **Xác thực (Authentication)**: Đảm bảo người dùng hoặc hệ thống đang cố gắng truy cập vào cơ sở dữ liệu là hợp lệ. Các phương pháp xác thực phổ biến bao gồm:
   - ***Tên người dùng và mật khẩu:*** Cấp quyền truy cập dựa trên tên người dùng và mật khẩu hợp lệ.
   - ***Chứng thực hai yếu tố (2FA):*** Bổ sung thêm một lớp bảo mật (*ví dụ: mã OTP*) ngoài mật khẩu.
   - ***Chứng thực qua token hoặc chứng chỉ số:*** Sử dụng các chứng chỉ số hoặc mã token để xác thực người dùng.
2. **Phân quyền truy cập (Authorization)**: Khi xác thực thành công, hệ thống kiểm tra quyền hạn của người dùng và phân quyền xác định ai được phép làm gì trên cơ sở dữ liệu.

- ***Role-based access control (RBAC - Phân quyền theo vai trò):*** Người dùng được gán các vai trò với các quyền hạn tương ứng. *Ví dụ: người quản trị, người dùng thông thường.*
- ***Mandatory Access Control (MAC - Cấp quyền dựa trên các cấp độ bảo mật cụ thể):*** Bí mật hay công khai.
- ***Discretionary Access Control (DAC):*** Người dùng có quyền kiểm soát tài nguyên của mình và có thể cấp quyền cho người dùng khác.

3. **Mã hóa dữ liệu (Data Encryption)**: Bảo vệ dữ liệu khỏi các cuộc tấn công từ bên ngoài hoặc bên trong bằng cách chuyển đổi dữ liệu thành dạng không đọc được nếu không có khóa giải mã, có thể được áp dụng ở nhiều cấp độ:
   - ***Mã hóa dữ liệu khi lưu trữ (Data at rest encryption):*** Mã hóa toàn bộ cơ sở dữ liệu hoặc các phần tử cụ thể như tệp tin, bảng.
   - ***Mã hóa dữ liệu khi truyền tải (Data in transit encryption):*** Sử dụng các giao thức như SSL/TLS để mã hóa dữ liệu khi nó được gửi qua mạng.
4. **Kiểm toán và theo dõi (Auditing and Monitoring)**
   - ***Kiểm toán:*** Lưu lại nhật ký (log) các thao tác truy cập, thay đổi dữ liệu, hoặc các hành động khác trên cơ sở dữ liệu. Kiểm toán giúp phát hiện các hành vi đáng ngờ hoặc vi phạm chính sách bảo mật.
   - ***Giám sát:*** Theo dõi hoạt động của cơ sở dữ liệu theo thời gian thực để phát hiện các hoạt động đáng ngờ như tấn công SQL Injection hay truy cập bất thường.
5. **Chính sách bảo mật và thực thi (Security Policies and Enforcement)**: Các quy định về việc truy cập và sử dụng dữ liệu, thường bao gồm:
   - ***Chính sách mật khẩu:*** Quy định độ phức tạp, thời hạn và việc thay đổi mật khẩu.
   - ***Chính sách cập nhật:*** Yêu cầu cập nhật định kỳ phần mềm cơ sở dữ liệu để vá các lỗ hổng bảo mật.
   - ***Chính sách sao lưu dữ liệu:*** Đảm bảo rằng dữ liệu được sao lưu định kỳ để phục hồi trong trường hợp có sự cố.
6. **Phòng chống tấn công (SQL Injection)**: Một dạng tấn công phổ biến vào cơ sở dữ liệu. Kẻ tấn công lợi dụng việc không kiểm tra đầu vào từ người dùng để chèn các câu lệnh SQL độc hại nhằm truy cập hoặc thay đổi dữ liệu. Giải pháp là sử dụng câu lệnh chuẩn bị (prepared statements), cơ chế ORM (Object-Relational Mapping), và kiểm tra kỹ càng các dữ liệu đầu vào.
7. **Sao lưu và khôi phục dữ liệu (Backup and Recovery)**: Thường xuyên và có kế hoạch khôi phục khi xảy ra sự cố là rất quan trọng trong việc đảm bảo tính toàn vẹn và sẵn sàng của dữ liệu. Nếu có lỗi, mất dữ liệu hoặc tấn công, các bản sao lưu có thể giúp khôi phục dữ liệu mà không bị tổn thất.
8. **Phân vùng bảo mật (Database Segmentation)**: Giới hạn phạm vi tấn công, các phần của cơ sở dữ liệu có mức độ bảo mật khác nhau có thể được tách ra và quản lý riêng biệt.
9. **Kiểm soát truy cập vật lý (Physical Access Control)**: Không chỉ bảo vệ dữ liệu bằng phần mềm, mà hệ thống cơ sở dữ liệu phải được bảo vệ vật lý. Các trung tâm dữ liệu phải có bảo mật vật lý (như camera, hệ thống kiểm soát truy cập, bảo vệ) để ngăn chặn người không phận sự tiếp cận trực tiếp với phần cứng chứa cơ sở dữ liệu.

3.4 Trực quan hóa dữ liệu

Trong cơ sở dữ liệu, **Data Visualization (Trực quan hóa Dữ liệu)** là quá trình sử dụng các yếu tố hình ảnh như đồ thị, biểu đồ hoặc bản đồ để trình bày dữ liệu. Quá trình này chuyển đổi dữ liệu phức tạp, có dung lượng lớn hoặc dữ liệu số thành hình ảnh trình bày trực quan có thể xử lý dễ dàng hơn.

Việc trực quan hóa dữ liệu tốt là loại bỏ được tình trạng nhiễu loạn dữ liệu, làm nổi bật những thông tin hữu ích và nói lên câu chuyện. Hỗ trợ đưa ra các quyết định chiến lược chính xác hơn, cải thiện chất lượng dịch vụ khách hàng và tăng mức độ tương tác của nhân viên.

Các công cụ trực quan hóa dữ liệu giúp cải thiện và tự động hóa quá trình giao tiếp bằng hình ảnh nhằm đảm bảo độ chính xác và chi tiết. Chúng ta có thể sử dụng những hình ảnh trình bày trực quan để trích xuất những thông tin chuyên sâu hữu ích từ dữ liệu thô. Quá trình trực quan hóa dữ liệu gồm 5 bước:

1. **Xác định mục tiêu (Determine the Objective):** Xác định các câu hỏi mà tập dữ liệu hiện có của chúng ta có thể trả lời, một mục tiêu rõ ràng giúp xác định loại:
   - Dữ liệu đang sử dụng.
   - Phân tích chúng ta thực hiện.
   - Các phương tiện trực quan dùng để thể hiện những phát hiện một cách hiệu quả.
2. **Thu thập dữ liệu (Data Collection)**: Liên quan đến việc xác định nguồn dữ liệu bên trong, bên ngoài và có sẵn các tập dữ liệu lớn trực tuyến để mua, sử dụng.
3. **Làm sạch dữ liệu (Data Visualization)**: Liên quan đến việc loại bỏ dữ liệu dư thừa, thực hiện các phép tính toán để phân tích thêm hoặc lọc và chuyển đổi dữ liệu để đáp ứng các tiêu chí của câu hỏi.
4. **Chọn phương tiện trực quan hóa dữ liệu (Select Data Visualization Method)**: Chọn trong nhiều loại biểu đồ khác nhau để khám phá theo cách trực quan hiệu quả, có hai loại hình trực quan hóa dữ liệu chính. Mối quan hệ giữa các điểm dữ liệu và thông tin chuyên sâu mà chúng ta muốn thể hiện sẽ xác định các biểu diễn đồ họa tốt nhất.
5. **Tạo phương tiện trực quan hóa dữ liệu (Create Data Visualization Method)**: Sử dụng các công cụ trực quan hóa dữ liệu nhập tập dữ liệu cuối cùng và tự động tạo ra các báo cáo theo yêu cầu. Một số nguyên tắc thiết kế để trực quan hóa dữ liệu hiệu quả:
   - Thu hút sự chú ý vào chi tiết quan trọng bằng kích cỡ, màu sắc, phông chữ và đồ họa.
   - Cung cấp ngữ cảnh cho dữ liệu bằng các dấu hiệu trực quan.
   - Chọn kiểu phối màu phù hợp.
   - Sử dụng tiêu đề giải thích để cung cấp thông tin chuyên sâu chính cho khán giả và tập trung vào đúng câu hỏi.
   - Thêm nhãn và số rõ ràng.

3.5 Index và tối ưu hóa truy vấn

3.5.1 Index

Index là cấu trúc dữ liệu đặc biệt giúp tăng tốc việc truy vấn trong cơ sở dữ liệu. Khi tạo chỉ mục trên một hoặc nhiều cột trong bảng, cơ sở dữ liệu sẽ tạo ra một bản sao nhỏ hơn của bảng đó, chứa các giá trị của các cột được chỉ mục và con trỏ đến các hàng tương ứng trong bảng chính. Các loại chỉ mục phổ biến:

1. **B-tree Index:** Phổ biến nhất, dựa theo cấu trúc của cây cân bằng, giúp tăng tốc các truy vấn tìm kiếm, lọc và sắp xếp dữ liệu. Cơ chế hoạt động cho phép truy xuất dữ liệu nhanh chóng bằng cách tìm kiếm nhị phân trong các mức của cây.
   Ví dụ: Muốn tăng tốc độ truy vấn dựa trên cột company_id, biết Bảng Company với các cột company_id, name, url, và company_size.

2. **Hash Index:** Dùng cho các phép toán tra cứu chính xác (dùng toán tử =), dựa trên cấu trúc dữ liệu Hash-Table, không hỗ trợ tốt các truy vấn phạm vi (như BETWEEN, >, <).

```
CREATE INDEX idx_company_name ON Company (name)
SELECT * FROM users WHERE name = 'IBM';
```

Listing 1: **Hash Index**: Thường xuyên tra cứu username để xác thực người dùng từ bảng Users.

3. **Full-Text Index:** Tối ưu hóa việc tìm kiếm văn bản tự do trong các trường văn bản lớn (*ví dụ: TEXT, VARCHAR*), sử dụng trong các trường hợp tìm kiếm theo nội dung tài liệu, bài viết hoặc các chuỗi văn bản dài.

```
CREATE FULL TEXT INDEX idx_posting_title ON Posting (title);
```

Listing 2: **Full-Text Index**: Tìm kiếm các tiêu đề chứa từ khóa phổ biến nhất từ bảng Posting.

4. **Unique Index:** Duy trì tính toàn vẹn dữ liệu, đảm bảo giá trị trong cột hoặc tập hợp các cột là duy nhất, ngăn ngừa việc thêm các bản ghi trùng lặp vào bảng.

```
CREATE UNIQUE INDEX idx_zip_code ON company_locations(zip_code);
```

Listing 3: **Unique Index**: Đảm bảo không có 2 nơi nào có cùng zip_code.

5. **Composite Index:** Chỉ mục được tạo trên nhiều cột trong bảng, thường dùng trong các trường hợp truy vấn kết hợp nhiều cột trong câu lệnh WHERE. Điều này giúp tối ưu hóa truy vấn nhiều điều kiện cùng lúc.

```
CREATE INDEX idx_apply_view_composite ON posting_state(apply_rate,
    views);
```

Listing 4: **Composite Index**: Truy vấn trên cột apply_rate và views từ bảng Posting_State.

6. **Clustered Index:** Một loại chỉ mục trong đó dữ liệu thực sự được lưu trữ và sắp xếp theo thứ tự của chỉ mục. Mỗi bảng chỉ có thể có một chỉ mục clustered, vì nó xác định cách dữ liệu được sắp xếp vật lý trên đĩa.
   - Ưu điểm:
   \+ Tăng tốc độ truy vấn.
   \+ Đảm bảo tính toàn vẹn dữ liệu.
   - Nhược điểm:
   \+ Giảm hiệu suất các thao tác ghi dữ liệu(INSERT, UPDATE, DELETE).
   \+ Việc duy trì và quản lý chỉ mục có thể gây tốn tài nguyên.
   \+ Không có lợi ích lớn khi sử dụng chỉ mục cho các bảng có kích thước nhỏ.

3.5.2 Tối ưu hóa truy vấn

Tối ưu hóa câu truy vấn giúp làm giảm thời gian thực thi của các câu lệnh SQL và tăng hiệu suất của hệ thống. Các kỹ thuật tối ưu hóa giúp cơ sở dữ liệu xử lý lượng lớn dữ liệu một cách hiệu quả hơn. Một số kỹ thuật tối ưu hóa câu truy vấn:

1. **Sử dụng index một cách hợp lý:** Tạo chỉ mục trên các cột thường xuyên được sử dụng trong điều kiện WHERE, JOIN, và ORDER BY. Đồng thời tránh chỉ mục trên các cột có nhiều giá trị trùng lặp, vì điều này có thể không mang lại hiệu suất tốt.
2. **Hạn chế SELECT tất cả các cột trong bảng:** Thay vì sử dụng SELECT *, chỉ nên truy vấn những cột cần thiết để tăng hiệu suất, tránh việc truy vấn dữ liệu dư thừa, giúp làm giảm khối lượng công việc mà cơ sở dữ liệu phải thực hiện khi lấy và trả về kết quả.
3. **Sử dụng các operation hiệu quả và đơn giản nhất có thể:** Các phép toán phức tạp hoặc không cần thiết nên được loại bỏ hoặc tối ưu hóa. Sử dụng các hàm tích hợp của SQL (như COUNT(), SUM(), AVG()) một cách hiệu quả sẽ giúp giảm thời gian xử lý.
4. **Tối ưu hóa các phép tính JOIN, ORDER BY và GROUP BY:** Các phép tính này thường tốn tài nguyên, do đó cần sử dụng chỉ mục để cải thiện hiệu suất. Đảm bảo rằng các cột được sử dụng trong các phép tính này đều có chỉ mục phù hợp.
5. **Partition (phân vùng dữ liệu):** Chia một bảng thành nhiều bảng nhỏ hơn, khi thực hiện truy vấn, hệ thống chỉ cần truy xuất phân vùng liên quan thay vì toàn bộ bảng, giúp câu truy vấn nhanh hơn.
6. **Caching (bộ nhớ đệm):** Lưu trữ dữ liệu thường xuyên truy vấn vào bộ nhớ đệm để có thể truy xuất nhanh khi cần, giúp giảm thiểu số lần truy vấn trực tiếp tới cơ sở dữ liệu, từ đó cải thiện hiệu suất hệ thống.
7. **Query Analysis (phân tích truy vấn):** Sử dụng các công cụ như EXPLAIN hoặc ANALYZE để kiểm tra các kế hoạch thực thi truy vấn. Điều này giúp phát hiện các vấn đề như quét bảng toàn bộ (full table scan), sử dụng chỉ mục không hiệu quả, hoặc các vấn đề khác mà hệ thống gặp phải.
8. **Tối ưu hóa bằng cách sử dụng View hoặc Materialized View:** Sử dụng View hoặc Materialized View để truy xuất nhanh các kết quả đã tính toán trước. Điều này giúp giảm thiểu việc tính toán lại các phép toán phức tạp trong mỗi truy vấn.
9. **Sử dụng các ràng buộc để kiểm soát dữ liệu:** Các ràng buộc như CHECK, FOREIGN KEY, hoặc UNIQUE không chỉ giúp duy trì tính toàn vẹn dữ liệu mà còn giảm thiểu số lần cần kiểm tra và xử lý lỗi sau khi truy vấn hoàn tất.

3.6 Cấu trúc lưu trữ

3.6.1 Heap Storage (Lưu trữ kiểu đống)

Heap Storage (Lưu trữ kiểu đống) là lưu trữ dữ liệu một cách ngẫu nhiên mà không có thứ tự cụ thể. Khi có một bản ghi mới, nó sẽ được thêm vào bất kỳ chỗ trống nào có sẵn.

- Ưu điểm:

+ Đơn giản trong việc chèn dữ liệu mới.

+ Phù hợp các tình huống khi dữ liệu được truy vấn bằng cách quét toàn bộ (full table scan).

- Nhược điểm: Tìm kiếm và truy vấn dữ liệu chậm hơn so với các cấu trúc lưu trữ có thứ tự vì phải quét toàn bộ bảng.

3.6.2 Sequential Storage (Lưu trữ tuần tự)

Sequential Storage (Lưu trữ tuần tự) là lưu trữ dữ liệu theo một thứ tự nhất định dựa trên một hoặc nhiều cột. Điều này cho phép truy vấn dữ liệu theo thứ tự được thực hiện nhanh hơn.

- Ứng dụng: Thường được sử dụng cho các bảng cần sắp xếp dữ liệu như bảng lưu trữ các giao dịch được sắp xếp theo thời gian.

- Ưu điểm: Tốc độ truy vấn và tìm kiếm nhanh khi cần truy vấn dữ liệu theo thứ tự.

- Nhược điểm: Chi phí cao trong việc chèn hoặc xóa bản ghi mới vì cần duy trì thứ tự.

3.6.3 Index Storage (Lưu trữ chỉ mục)

Index Storage (Lưu trữ chỉ mục) là cấu trúc dữ liệu đặc biệt giúp tăng tốc độ truy vấn dữ liệu bằng cách tạo ra một bảng tra cứu cho các giá trị của một hoặc nhiều cột trong bảng dữ liệu.

- Các loại chỉ mục phổ biến:

+ ***B-Tree Index:*** Tạo ra cấu trúc cây với các nút được sắp xếp để tìm kiếm và duyệt nhanh chóng. Phù hợp cho các truy vấn có điều kiện range (phạm vi).

+ ***Hash Index:*** Sử dụng hàm băm để xác định vị trí của các bản ghi. Tốt cho các truy vấn tìm kiếm giá trị chính xác nhưng không hỗ trợ tìm kiếm phạm vi.

- Ưu điểm: Tăng tốc độ truy vấn.

- Nhược điểm: Chi phí tạo và bảo trì chỉ mục, đặc biệt khi chèn hoặc xóa dữ liệu.

3.6.4 Columnar Storage (Lưu trữ theo cột)

Columnar Storage (Lưu trữ theo cột) là lưu trữ dữ liệu theo từng cột thay vì từng hàng như trong các cơ sở dữ liệu quan hệ truyền thống. Mỗi cột của bảng được lưu trữ riêng biệt.

- Ứng dụng: Thường được sử dụng trong các hệ thống kho dữ liệu (data warehouses) và phân tích dữ liệu lớn (big data) vì khả năng nén dữ liệu và truy xuất nhanh.

- Ưu điểm:

+ Tốc độ truy vấn cao với các phép tính tổng hợp (aggregation) như SUM, AVG.

+ Khả năng nén dữ liệu tốt hơn.

- Nhược điểm: Hiệu suất thấp hơn khi thực hiện các thao tác chèn hoặc cập nhật từng bản ghi.

3.6.5 Row-Oriented Storage (Lưu trữ theo hàng)

Row-Oriented Storage (Lưu trữ theo hàng) là phương pháp lưu trữ truyền thống, trong đó các bản ghi được lưu theo hàng và mỗi hàng chứa dữ liệu tất cả các cột của bản ghi đó.

- Ứng dụng: Phù hợp cho các ứng dụng OLTP (Online Transaction Processing) khi mà dữ liệu được truy cập theo từng bản ghi riêng lẻ.

- Ưu điểm:

+ Thao tác chèn, cập nhật và xóa nhanh.

+ Dễ dàng cho các truy vấn lấy tất cả thông tin của một bản ghi.

- Nhược điểm: Kém hiệu quả khi thực hiện các truy vấn cần truy cập một số ít cột trên nhiều hàng.

3.6.6 Object Storage (Lưu trữ đối tượng)

Object Storage (Lưu trữ đối tượng) là lưu trữ dữ liệu dưới dạng các đối tượng (objects), mỗi đối tượng bao gồm dữ liệu, siêu dữ liệu, và một ID duy nhất để nhận diện.

- Ứng dụng: Lưu trữ các tệp lớn như hình ảnh, video và dữ liệu phi cấu trúc trong các hệ thống đám mây.

- Ưu điểm:

+ Khả năng mở rộng lớn, phù hợp với lưu trữ phi cấu trúc.

+ Tích hợp tốt với các dịch vụ đám mây.

- Nhược điểm: Không phù hợp cho các truy vấn truyền thống như trong các hệ cơ sở dữ liệu quan hệ.

3.6.7 File-based Storage (Lưu trữ dựa trên tệp)

File-based Storage (Lưu trữ dựa trên tệp) là lưu trữ dữ liệu dưới dạng các tệp (files) trong hệ thống tệp (file system) của máy tính. Thường được sử dụng khi không cần các tính năng của hệ thống cơ sở dữ liệu phức tạp.

- Ứng dụng: Thích hợp cho các ứng dụng cần lưu trữ đơn giản như nhật ký (log), lưu trữ các cấu hình.

- Ưu điểm:

+ Dễ triển khai và quản lý.

+ Không yêu cầu phần mềm DBMS.

- Nhược điểm: Thiếu tính năng truy vấn và bảo mật như trong các DBMS.

3.6.8 In-Memory Storage (Lưu trữ trong bộ nhớ)

In-Memory Storage (Lưu trữ trong bộ nhớ) là lưu trữ dữ liệu được lưu trữ trực tiếp trong bộ nhớ RAM thay vì lưu trên ổ đĩa. Điều này cho phép truy cập và xử lý dữ liệu với tốc độ rất cao.

- Ứng dụng: Yêu cầu hiệu suất cao và thời gian truy cập thấp như các ứng dụng giao dịch tài chính, hệ thống caching.

- Ưu điểm: Tốc độ truy vấn và xử lý dữ liệu nhanh chóng.

- Nhược điểm:

+ Khả năng lưu trữ bị giới hạn bởi dung lượng RAM.

+ Dữ liệu có nguy cơ bị mất nếu hệ thống gặp sự cố mà không được sao lưu.

3.6.9 Distributed Storage (Lưu trữ phân tán)

Distributed Storage (Lưu trữ phân tán) là lưu trữ dữ liệu trên nhiều máy chủ hoặc nhiều khu vực khác nhau để đảm bảo tính sẵn sàng cao và khả năng mở rộng. Các hệ thống lưu trữ phân tán có thể chia nhỏ và lưu trữ dữ liệu trên nhiều nút (nodes).

- Ứng dụng: Phù hợp cho các hệ thống lớn như big data, điện toán đám mây và các ứng dụng yêu cầu độ sẵn sàng và tính toàn vẹn dữ liệu cao.

- Ưu điểm:

+ Khả năng mở rộng và sẵn sàng cao.

+ Đảm bảo tính liên tục khi một hoặc nhiều nút gặp sự cố.

- Nhược điểm:

+ Phức tạp trong việc quản lý và bảo trì.

+ Yêu cầu kết nối mạng ổn định và nhanh chóng.

3.7 Các check constraints

1. `salary` **Constraints**: Đảm bảo rằng mức lương tối thiểu không lớn hơn mức lương tối đa và các mức lương là số dương.

```
SELECT * FROM Salary WHERE pay_period IS NOT NULL
    AND UPPER(pay_period) NOT IN ('HOURLY', 'WEEKLY', 'MONTHLY',
        'YEARLY');
```

Listing 5: Trước kiểm thử **Constraint 1.** `salary`.

	salary_id	job_id	currency	compensation_type	pay_period
1	11760	3895598632	USD	BASE_SALARY	BIWEEKLY
2	17734	3904387609	USD	BASE_SALARY	BIWEEKLY
3	18105	3904388204	USD	BASE_SALARY	BIWEEKLY
4	19961	3904386637	USD	BASE_SALARY	BIWEEKLY
5	21878	3904996213	USD	BASE_SALARY	BIWEEKLY
6	22553	3904995353	USD	BASE_SALARY	BIWEEKLY
7	24005	3904991870	USD	BASE_SALARY	BIWEEKLY
8	29798	3905333190	USD	BASE_SALARY	BIWEEKLY
9	39419	3906252426	USD	BASE_SALARY	BIWEEKLY
10	40786	1218575			
11	40787	9615617			
12	40788	11009123			
13	40789	56482768			

Hình 3: Trước kiểm thử **Constraint 1.** `salary`.

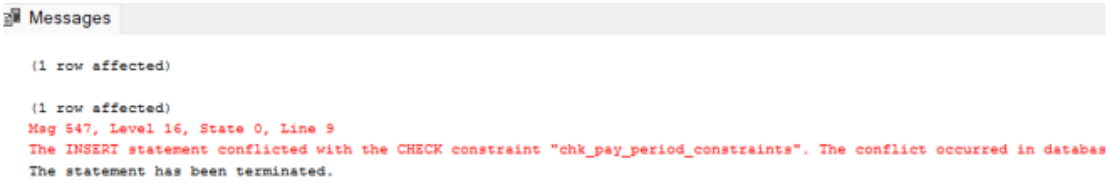
```
UPDATE Salary
SET pay_period = NULL
WHERE pay_period = ' ';
UPDATE Salary
SET pay_period = 'WEEKLY'
WHERE UPPER(pay_period) = 'BIWEEKLY';

ALTER TABLE Salary
ADD CONSTRAINT chk_pay_period_constraints
CHECK (
    pay_period IS NULL OR UPPER(pay_period) IN ('HOURLY', 'WEEKLY
        ', 'MONTHLY', 'YEARLY')
);
```

Listing 6: Lệnh tạo **Constraint 1.** `salary` chuyển chuỗi rỗng thành NULL và BIWEEKLY thành WEEKLY.

```
INSERT INTO Salary (job_id, currency, compensation_type,
    pay_period)
VALUES (1, 'USD', 'Full-time', 'WEEKLY');
INSERT INTO Salary (job_id, currency, compensation_type,
    pay_period)
```

```
    VALUES (2, 'USD', 'Part-time', 'NULL');
    INSERT INTO Salary (job_id, currency, compensation_type,
        pay_period)
    VALUES (3, 'USD', 'Full-time', 'DAILY');
```

Listing 7: Kiểm thử **Constraint 1.** `salary`.



Hình 4: Kiểm thử **Constraint 1.** `salary`.

2. `application_url` **Constraints**: Dữ liệu thêm hoặc cập nhật trong bảng Additional_info phải có dạng 'http://' hoặc 'https://'.

```
    ALTER TABLE Additional_Info
    ADD CONSTRAINT chk_application_url_format
    CHECK (application_url LIKE 'http://%' OR application_url LIKE '
        https://%');
```

Listing 8: Lệnh tạo **Constraint 2.** `application_url`.

3. `follower_count` **Constraints**: Đảm bảo rằng follower_count trong bẳng Emloyee_Count phải luôn lớn hơn hoặc bằng 0.

```
    ALTER TABLE Employee_Count
    ADD CONSTRAINT chk_follower_count_non_negative
    CHECK (follower_count >= 0);
```

Listing 9: Lệnh tạo **Constraint 3.** `follower_count`.

4. `zip_code` **Constraints**: Đảm bảo rằng mã ZIP (zipcode) trong bảng Company_Location có độ dài từ 5 đến 10 ký tự.

```
    ALTER TABLE Company_Location
    ADD CONSTRAINT chk_zipcode_length
    CHECK (LEN(zipcode) BETWEEN 5 AND 10);
```

Listing 10: Lệnh tạo **Constraint 3.** `follower_count`.

3.8 Các trigger liên quan

1. **Trigger for** `closed_time`: Tự động cập nhật thời gian khi công việc bị đánh dấu đóng, trigger này sẽ tự động cập nhật cột `closed_time` nếu nó chưa có giá trị.

```
ALTER TABLE Additional_Info
DROP COLUMN close_time;
ALTER TABLE Additional_Info
ADD close_time DATETIME NULL;

CREATE TRIGGER trg_UpdateClosedTime
ON Additional_Info
AFTER UPDATE
AS
BEGIN
    UPDATE Additional_Info
    SET closed_time = GETDATE()
    WHERE additional_info_id IN (SELECT additional_info_id FROM
        inserted)
          AND closed_time IS NULL;
END;
```

Listing 11: Lệnh tạo **Trigger 2. For** `closed_time`.

```
SELECT * FROM Additional_Info WHERE additional_info_id = 111170;
```

Listing 12: Kiểm thử **Trigger 2. For** `closed_time`.

	additional_info_id	formatted_experience_level	posting_domain	application_url	posting_id	close_time
1	111170	Mid Level	NULL	https://www.glassdoor.com/jobs/view/438291/	6	2025-01-04 17:57:28.490

Hình 5: Kiểm thử **Trigger 2. For** `closed_time`.

2. **Trigger to validate** `salary`: Khi thêm hoặc cập nhật thông tin về mức lương, trigger này sẽ đảm bảo rằng mức lương tối thiểu không thể cao hơn mức lương tối đa.

```
CREATE TRIGGER trg_check_salary_value
ON Salary_Type
AFTER INSERT
AS
BEGIN
    SET NOCOUNT ON;
    IF EXISTS (
        SELECT 1
        FROM INSERTED
        WHERE value > 1000000
    )
    BEGIN
        PRINT 'Warning: Muc luong da vuot qua 1,000,000 xin vui long
            kiem tra lai';
    END
END;
```

Listing 13: Lệnh tạo **Trigger 3. To validate** `salary`.

```
INSERT INTO Salary_Type (salary_id, salary_type, value)
VALUES (1, 'min', 900000);
```

```
SELECT * FROM Salary_Type;
```

Listing 14: Thêm bản ghi mới hợp lệ vào bảng Salary_Type để kiểm tra.

	salary_type	salary_id	value
1	Base	1	900000
2	med	1	20
3	Min	1	900000
4	max	2	25
5	min	2	23
6	max	3	120000
7	min	3	100000
8	max	4	200000
9	min	4	10000
10	max	5	35

Hình 6: Kiểm thử **Trigger 3. To validate** `salary` hợp lệ.

```
INSERT INTO Salary_Type (salary_id, salary_type, value)
VALUES (1, 'min', 900000);
```

Listing 15: Thêm bản ghi có giá trị value vượt quá 1,000,000 vào bảng Salary_Type để kiểm tra.

```
Warning: Muc luong da vuot qua 1,000,000 xin vui long kiem tra lai

(1 row affected)
```

Hình 7: Kiểm thử **Trigger 3. To validate** `salary` không hợp lệ.

3. **Trigger to auto-update** `follower_count`: Khi một công ty có thêm người theo dõi mới, trigger này sẽ tự động cập nhật cột `follower_count`.

```
CREATE TRIGGER trg_UpdateFollowerCount
ON Employee_Count
AFTER INSERT, UPDATE
AS
BEGIN
    UPDATE Employee_Count
    SET follower_count = follower_count + 1
    WHERE company_id IN (SELECT company_id FROM inserted);
END;
```

Listing 16: Lệnh tạo **Trigger 3. to auto-update** `follower_count`.

4. **Trigger delete related records when** deleting `job_id`: Khi một công việc bị xóa khỏi bảng Job, tự động xóa các bản ghi liên quan trong bảng Job_Has_Benefit và Job_Has_Industry.

```sql
CREATE TRIGGER trg_delete_job_benefits
ON Job
AFTER DELETE
AS
BEGIN
    SET NOCOUNT ON;
    DELETE FROM Job_Has_Benefit
    WHERE job_id IN (SELECT job_id FROM DELETED);
    DELETE FROM Job_Has_Industry
    WHERE job_id IN (SELECT job_id FROM DELETED);
END;
```

Listing 17: Lệnh tạo **Trigger 4. Delete related records when deleting** `job_id`.

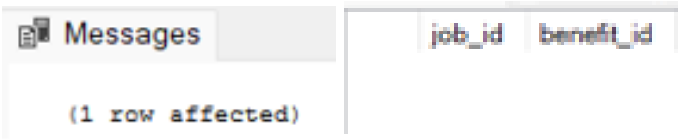
```sql
SELECT * FROM Job WHERE job_id = 173586092;
SELECT * FROM Job_Has_Benefit WHERE job_id = 173586092;
```

Listing 18: Trước khi kiểm thử **Trigger 4. Delete related records when deleting** `job_id`.

Hình 8: Trước khi kiểm thử **Trigger 4. Delete related records when deleting** `job_id`.

```sql
DELETE FROM Job WHERE job_id = 173586092;

SELECT * FROM Job_Has_Benefit WHERE job_id = 173586092;
```

Listing 19: **Kiểm thử Trigger 4. Delete related records when deleting** `job_id`.



Hình 9: Kiểm thử **Trigger 4. Delete related records when deleting** `job_id`.

4 Xử lý các dữ liệu mẫu để import vào CSDL trên DBMS

4.1 Các câu lệnh tạo bảng

1. **Bảng Company:**

```
CREATE TABLE Company (
    company_id BIGINT PRIMARY KEY,
    company_size SMALLINT,
    description TEXT,
    company_name VARCHAR(255),
    company_url VARCHAR(255)
);
```

Listing 20: Lệnh tạo Bảng **Company**

2. **Bảng Job:**

```
CREATE TABLE Job (
    job_id BIGINT PRIMARY KEY,
    job_description TEXT
);
```

Listing 21: Lệnh tạo Bảng **Job**

3. **Bảng Skill:**

```
CREATE TABLE Skill (
    skill_abr VARCHAR(10) PRIMARY KEY,
    skill_name VARCHAR(255),
    category VARCHAR(100)
);
```

Listing 22: Lệnh tạo Bảng **Skill**

4. **Bảng Benefit:**

```
CREATE TABLE Benefit (
    benefit_id INT IDENTITY(1,1) PRIMARY KEY,
    inferred VARCHAR(255)
);
```

Listing 23: Lệnh tạo Bảng **Benefit**

5. **Bảng Benefit_Type:**

```
CREATE TABLE Benefit_Type (
    benefit_type_id INT IDENTITY(1,1) PRIMARY KEY,
    type VARCHAR(100)
);
```

Listing 24: Lệnh tạo Bảng **Benefit_Type**

6. **Bảng Company_Industry:**

```
CREATE TABLE Company_Industry (
    industry_id INT PRIMARY KEY,
    industry_name VARCHAR(255)
);
```

Listing 25: Lệnh tạo Bảng **Company_Industry**

7. **Bảng Posting:**

```
CREATE TABLE Posting (
    posting_id INT PRIMARY KEY,
    title VARCHAR(255),
    posting_description TEXT,
    job_posting_url VARCHAR(255),
    application_type VARCHAR(100),
    skills_description TEXT,
    formatted_worktype VARCHAR(100),
    zip_code VARCHAR(255),
    remote_allowed BIT,
    location VARCHAR(255),
    company_id BIGINT,
    job_id BIGINT
);
```

Listing 26: Lệnh tạo Bảng **Posting**

8. **Bảng Posting_State:**

```
CREATE TABLE Posting_State (
    posting_state_id INT IDENTITY(1,1) PRIMARY KEY,
    original_listed_time DATETIME,
    listed_time DATETIME,
    applies INT,
    views INT,
    expiry DATETIME,
    posting_id INT,
    apply_rate FLOAT,
    remaining_time INT
);
```

Listing 27: Lệnh tạo Bảng **Posting_State**

9. **Bảng Additional_Info:**

```
CREATE TABLE Additional_Info (
    additional_info_id INT IDENTITY(1,1) PRIMARY KEY,
    formatted_experience_level VARCHAR(100),
    posting_domain VARCHAR(255),
    application_url VARCHAR(1000),
    closed_time DATETIME,
    posting_id INT);
```

Listing 28: Lệnh tạo Bảng **Additional_Info**

10. **Bảng Salary:**

```
CREATE TABLE Salary (
    salary_id INT PRIMARY KEY,
    job_id BIGINT,
    currency VARCHAR(10),
    compensation_type VARCHAR(50),
    pay_period VARCHAR(50)
);
```

Listing 29: Lệnh tạo Bảng **Salary**

11. **Bảng Salary_Type:**

```
CREATE TABLE Salary_Type (
    salary_type VARCHAR(10),
    salary_id INT,
    value FLOAT,
    PRIMARY KEY (salary_id, salary_type)
);
```

Listing 30: Lệnh tạo Bảng **Salary_Type**

12. **Bảng Requires:**

```
CREATE TABLE Requires (
    skill_abr VARCHAR(10),
    job_id BIGINT,
    PRIMARY KEY (skill_abr, job_id)
);
```

Listing 31: Lệnh tạo Bảng **Requires**

13. **Bảng Job_Has_Benefit:**

```
CREATE TABLE Job_Has_Benefit (
    job_id BIGINT,
    benefit_id INT,
    PRIMARY KEY (job_id, benefit_id)
);
```

Listing 32: Lệnh tạo Bảng **Job_Has_Benefit**

14. **Benefit_Has_Benefit_Type:**

```
CREATE TABLE Benefit_Has_Benefit_Type (
    benefit_id INT,
    benefit_type_id INT,
    PRIMARY KEY (benefit_id, benefit_type_id)
);
```

Listing 33: Lệnh tạo Bảng **Benefit_Has_Benefit_Type**

15. **Job_Has_Industry:**

```
CREATE TABLE Job_Has_Industry (
    job_id BIGINT,
    industry_id INT,
    PRIMARY KEY (job_id, industry_id)
);
```

Listing 34: Lệnh tạo Bảng **Job_Has_Industry**

16. **Company_Has_Industry:**

```
CREATE TABLE Company_Has_Industry (
    company_id BIGINT,
    industry_id INT,
    PRIMARY KEY (company_id, industry_id)
);
```

Listing 35: Lệnh tạo Bảng **Company_Has_Industry**

17. **Company_Speciality:**

```
CREATE TABLE Company_Speciality (
    speciality_id INT IDENTITY(1,1) PRIMARY KEY,
    speciality VARCHAR(255)
);
```

Listing 36: Lệnh tạo Bảng **Company_Speciality**

18. **Company_Has_Speciality:**

```
CREATE TABLE Company_Has_Speciality (
    company_id BIGINT,
    speciality_id INT,
    PRIMARY KEY (company_id, speciality_id)
);
```

Listing 37: Lệnh tạo Bảng **Company_Has_Speciality**

19. **Employee_Count:**

```
CREATE TABLE Employee_Count (
    time_recorded DATETIME,
    company_id BIGINT,
    employee_count INT,
    follower_count INT,
    PRIMARY KEY (time_recorded, company_id)
);
```

Listing 38: Lệnh tạo Bảng **Employee_Count**

20. **Company_Location:**

```
CREATE TABLE Company_Location (
    location_id INT IDENTITY(1,1) PRIMARY KEY,
    zipcode VARCHAR(255),
    number VARCHAR(10),
    street VARCHAR(255),
    state VARCHAR(100),
    country VARCHAR(100),
    city VARCHAR(255),
    company_id BIGINT
);
```

Listing 39: Lệnh tạo Bảng **Company_Location**

4.2 Các ràng buộc khoá ngoại trên các bảng

```
-- Posting table
ALTER TABLE Posting
ADD FOREIGN KEY (company_id) REFERENCES Company(company_id);

ALTER TABLE Posting
ADD FOREIGN KEY (job_id) REFERENCES Job(job_id);

-- Posting_State table
ALTER TABLE Posting_State
ADD CONSTRAINT FK_PostingState_postingStateID FOREIGN KEY (
    posting_state_id) REFERENCES Posting(posting_id);

-- Additional_Info table
ALTER TABLE Additional_Info
ADD CONSTRAINT FK_Additional_Info_Additional_InfoID FOREIGN KEY (
    additional_info_id) REFERENCES Posting(posting_id);

-- Salary table
ALTER TABLE Salary
ADD FOREIGN KEY (job_id) REFERENCES Job(job_id);

-- Salary_Type table
ALTER TABLE Salary_Type
ADD FOREIGN KEY (salary_id) REFERENCES Salary(salary_id);

-- Requires table
ALTER TABLE Requires
ADD FOREIGN KEY (skill_abr) REFERENCES Skill(skill_abr);

ALTER TABLE Requires
ADD FOREIGN KEY (job_id) REFERENCES Job(job_id);

-- Job_Has_Benefit table
ALTER TABLE Job_Has_Benefit
ADD FOREIGN KEY (job_id) REFERENCES Job(job_id);

ALTER TABLE Job_Has_Benefit
```

```
ADD FOREIGN KEY (benefit_id) REFERENCES Benefit(benefit_id);

-- Benefit_Has_Benefit_Type table
ALTER TABLE Benefit_Has_Benefit_Type
ADD FOREIGN KEY (benefit_id) REFERENCES Benefit(benefit_id);

ALTER TABLE Benefit_Has_Benefit_Type
ADD FOREIGN KEY (benefit_type_id) REFERENCES Benefit_Type(benefit_type_id
    );

-- Job_Has_Industry table
ALTER TABLE Job_Has_Industry
ADD FOREIGN KEY (job_id) REFERENCES Job(job_id);

ALTER TABLE Job_Has_Industry
ADD FOREIGN KEY (industry_id) REFERENCES Company_Industry(industry_id);

-- Company_Has_Industry table
ALTER TABLE Company_Has_Industry
ADD FOREIGN KEY (company_id) REFERENCES Company(company_id);

ALTER TABLE Company_Has_Industry
ADD FOREIGN KEY (industry_id) REFERENCES Company_Industry(industry_id);

-- Company_Has_Speciality table
ALTER TABLE Company_Has_Speciality
ADD FOREIGN KEY (company_id) REFERENCES Company(company_id);

ALTER TABLE Company_Has_Speciality
ADD FOREIGN KEY (speciality_id) REFERENCES Company_Speciality(
    speciality_id);

-- Employee_Count table
ALTER TABLE Employee_Count
ADD FOREIGN KEY (company_id) REFERENCES Company(company_id);

-- Company_Location table
ALTER TABLE Company_Location
ADD FOREIGN KEY (company_id) REFERENCES Company(company_id);
```

Listing 40: Các ràng buộc khoá ngoại trên các bảng.

4.3 Xử lý dữ liệu mẫu và import vào CSDL trên DBMS

1. **Thiết lập các module**

```
import pandas as pd
import numpy as np
import re
from sqlalchemy import create_engine
```

Listing 41: Thiết lập các module.

2. **Lệnh gọi các hàm**

```
Company()
Company_Has_Industry()
Company_Industry()
Company_Location()
Company_Has_Speciality()
Company_Speciality()
Employee_Count()
Job_Has_Industry()
Posting()
Posting_State()
Additional_Info()
Job()
Skill()
Benefit()
Benefit_Type()
Benefit_Has_Benefit_Type()
Job_Has_Benefit()
Requires()
Salary()
Salary_Type()
```

Listing 42: Lệnh gọi các hàm.

3. **Hàm kết nối cơ sở dữ liệu**

```
connection_string = "mssql+pyodbc://ZRMR/CO3127?driver=ODBC+Driver
    +17+for+SQL+Server"
engine = create_engine(connection_string)
```

Listing 43: Hàm kết nối cơ sở dữ liệu.

4. **Hàm làm sạch dữ liệu**

```
def clean_text(text):
    if isinstance(text, str):
        text = re.sub(r'[^\x00-\x7F]+', '', text)
    return text
```

Listing 44: Hàm làm sạch dữ liệu.

5. **Hàm chuyển đổi thời gian TIMESTAMP thành DATETIME**

```
def safe_convert_timestamp(timestamp):
    try:
        if timestamp > 10000000000:
            return pd.to_datetime(timestamp, unit='ms')
        else:
            return pd.to_datetime(timestamp, unit='s')
    except Exception as e:
        print(f"Error with timestamp: {timestamp}, Error: {e}")
        return pd.NaT
```

Listing 45: Hàm chuyển đổi thời gian TIMESTAMP thành DATETIME.

6. **Các lệnh đọc CSV file và chèn dữ liệu**

- **Bảng Company**:

```
def Company():
    companies = pd.read_csv('companies/companies.csv')
    attribute = ['company_id', 'company_size', 'description', 'name',
        'url']
    Company = companies[attribute].copy()
    for col in attribute:
        Company[col] = Company[col].apply(clean_text)
    Company.rename(columns={
        'name': 'company_name',
        'url': 'company_url'
    }, inplace=True)
    try:
        Company.to_sql('Company', con=engine, if_exists='append',
            index=False)
        print("Company has been inserted successfully!")
    except Exception as e:
        print(f"Company: An error occurred during insertion: {e}")
```

Listing 46: Các lệnh đọc và chèn dữ liệu vào bảng **Company**.

	company_id	company_size	description	company_name	company_url
1	1009	7	At IBM, we do more than work. We create. We create...	IBM	https://www.linkedin.com/company/ibm
2	1016	7	Every day millions of people feel the impact of our int...	GE HealthCare	https://www.linkedin.com/company/gehealthcare
3	1025	7	Official LinkedIn of Hewlett Packard Enterprise, the g...	Hewlett Packard Enterprise	https://www.linkedin.com/company/hewlett-packard...
4	1028	7	Were a cloud technology company that provides orga...	Oracle	https://www.linkedin.com/company/oracle
5	1033	7	Accenture is a leading global professional services co...	Accenture	https://www.linkedin.com/company/accenture
6	1035	7	Every company has a mission. What's ours? To empo...	Microsoft	https://www.linkedin.com/company/microsoft
7	1038	7	Deloitte drives progress. Our firms around the world h...	Deloitte	https://www.linkedin.com/company/deloitte
8	1043	7	Siemens AG (Berlin and Munich) is a leading technol...	Siemens	https://www.linkedin.com/company/siemens
9	1044	7	At PwC, our purpose is to build trust in society and so...	PwC	https://www.linkedin.com/company/pwc
10	1052	7	We understand that our customers want an easier, le...	AT&T	https://www.linkedin.com/company/att

Hình 10: Bảng **Company**.

- **Bảng Job**:

```
def Job():
    job = pd.read_csv('jobs/job_industries.csv')
    Job = job[['job_id']].drop_duplicates()
    Job['job_description'] = ""
    try:
        Job.to_sql('Job', con=engine, if_exists='append', index=False
            )
        print("Job has been inserted successfully!")
    except Exception as e:
        print(f"Job: An error occurred during insertion: {e}")
```

Listing 47: Các lệnh đọc và chèn dữ liệu vào bảng **Job**.

	job_id	job_description
1	921716	
2	1218575	
3	2264355	
4	9615617	
5	10998357	
6	11009123	
7	23221523	
8	35982263	
9	56482768	
10	56924323	

Hình 11: Bảng **Job**.

- **Bảng Skill**:

```
def Skill():
    skills = pd.read_csv('mappings/skills.csv')
    skills['category'] = ""
    try:
        skills.to_sql('Skill', con=engine, if_exists='append', index=
            False)
        print("skills has been inserted successfully!")
    except Exception as e:
        print(f"skills: An error occurred during insertion: {e}")
```

Listing 48: Các lệnh đọc và chèn dữ liệu vào bảng **Skill**.

	skill_abr	skill_name	category
1	ACCT	Accounting/Auditing	
2	ADM	Administrative	
3	ADVR	Advertising	
4	ANLS	Analyst	
5	ART	Art/Creative	
6	BD	Business Development	
7	CNSL	Consulting	
8	CUST	Customer Service	
9	DIST	Distribution	
10	DSGN	Design	

Hình 12: Bảng **Skill**.

- **Bảng Benefit**:

```
def Benefit():
    benefit = pd.read_csv('jobs/benefits.csv')
    Benefit = benefit[['inferred']].copy()
    Benefit = Benefit[['inferred']]
    try:
        Benefit.to_sql('Benefit', con=engine, if_exists='append',
            index=False)
```

```
        print("Benefit has been inserted successfully!")
    except Exception as e:
        print(f"Benefit: An error occurred during insertion: {e}")
```

Listing 49: Các lệnh đọc và chèn dữ liệu vào bảng **Benefit**.

	benefit_id	inferred
1	1	0
2	2	0
3	3	0
4	4	0
5	5	0
6	6	0
7	7	0
8	8	0
9	9	0
10	10	0

Hình 13: Bảng **Benefit**.

- **Bảng Benefit_Type**:

```
def Benefit_Type():
    benefits = pd.read_csv('jobs/benefits.csv')
    Benefit_Type = benefits[['type']].drop_duplicates()
    Benefit_Type = Benefit_Type[['type']]
    try:
        Benefit_Type.to_sql('Benefit_Type', con=engine, if_exists='
            append', index=False)
        print("Benefit has been inserted successfully")
    except Exception as e:
        print(f'Benefit Type: An error occurred during insertion: {e}
            ')
```

Listing 50: Các lệnh đọc và chèn dữ liệu vào bảng **Benefit_Type**.

	benefit_type_id	type
1	1	Medical insurance
2	2	Vision insurance
3	3	Dental insurance
4	4	401(k)
5	5	Student loan assistance
6	6	Tuition assistance
7	7	Disability insurance
8	8	Paid maternity leave
9	9	Paid paternity leave
10	10	Child care support

Hình 14: Bảng **Benefit_Type**.

- **Bảng Company_Industry**:

```
def Company_Industry():
    industries = pd.read_csv('mappings/industries.csv')

    try:
        industries.to_sql('Company_Industry', con=engine, if_exists='
            append', index=False)
        print("Company_Industry has been inserted successfully!")
    except Exception as e:
        print(f"Company_Industry: An error occurred during insertion:
             {e}")
```

Listing 51: Các lệnh đọc và chèn dữ liệu vào bảng **Company_Industry**.

	industry_id	industry_name
1	1	Defense and Space Manufacturing
2	3	Computer Hardware Manufacturing
3	4	Software Development
4	5	Computer Networking Products
5	6	Technology, Information and Internet
6	7	Semiconductor Manufacturing
7	8	Telecommunications
8	9	Law Practice
9	10	Legal Services
10	11	Business Consulting and Services

Hình 15: Bảng **Company_Industry**.

- **Bảng Posting**:

```
def Posting():
    postings = pd.read_csv('postings.csv')
    Posting = postings[['title', 'description', 'job_posting_url', '
        application_type',
                        'skills_desc', 'formatted_work_type', '
                            zip_code', 'remote_allowed', 'location',
                             'company_id', 'job_id']].copy()
    Posting.rename(columns={
        'description': 'posting_description',
        'skills_desc': 'skills_description',
        'formatted_work_type': 'formatted_worktype'
    }, inplace = True)
    Posting['title'] = Posting['title'].replace({
        r'\s*/\s*': ' or ',
        r'\s*&\s*': ' and ',
    }, regex=True)
    Posting['title'] = Posting['title'].replace({
        'Part - Time': 'Part-Time',
        'Part- Time': 'Part-Time',
```

```
        'Part -Time': 'Part-Time'
    })
    Posting['title'] = Posting['title'].replace({
        ' - ': ', ',
        '-': ', ',
        ' -': ', ',
        '- ': ', '
    })
    Posting['title'] = Posting['title'].str.strip()
    Posting['remote_allowed'] = Posting['remote_allowed'].fillna(0).
        astype(int)
    Posting['posting_id'] = range(1, len(Posting) + 1)
    column_order = ["posting_id", "title", "posting_description", "
        job_posting_url",
                "application_type", "skills_description", "
                    formatted_worktype",
                "zip_code", "remote_allowed", "location", "company_id
                    ", "job_id"]
    Posting = Posting[column_order]
    try:
        Posting.to_sql('Posting', con=engine, if_exists='append',
            index=False)
        print("Posting has been inserted successfully!")
    except Exception as e:
        print(f"Posting: An error occurred during insertion: {e}")
```

Listing 52: Các lệnh đọc và chèn dữ liệu vào bảng **Posting**.

	posting_id	title	posting_description	job_posting_url	application_type	skills_description	formatted_worktype	zip_code	remote_allowed	location	company_id	job_id
1	1	Marke...	Job descriptionA l...	https://www.li...	ComplexOnsit...	Requirements: ...	Full-time	8540	0	Princ...	2774458	921716
2	2	Ment...	At Aspen Therap...	https://www.li...	ComplexOnsit...	NULL	Full-time	80521	0	Fort ...	NULL	1829192
3	3	Assita...	The National Exe...	https://www.li...	ComplexOnsit...	We are currentl...	Full-time	45202	0	Cinci...	64896719	10998357
4	4	Senio...	Senior Associate ...	https://www.li...	ComplexOnsit...	This position re...	Full-time	11040	0	New ...	766262	23221523
5	5	Servic...	Looking for HVA...	https://www.li...	ComplexOnsit...	NULL	Full-time	52601	0	Burlin...	NULL	35982263
6	6	Econ...	Job summary:The...	https://www.li...	ComplexOnsit...	NULL	Internship	27601	0	Ralei...	1481176	91700727
7	7	Produ...	Company Descrip...	https://www.li...	SimpleOnsite...	NULL	Contract	NULL	1	Unite...	81942316	103254301
8	8	Buildi...	Summary: Due to...	https://www.li...	ComplexOnsit...	NULL	Full-time	94101	0	San ...	NULL	112576855
9	9	Respi...	At Children's, the ...	https://www.li...	OffsiteApply	• Requires the ...	Full-time	68102	0	Oma...	721189	1218575
10	10	Wors...	It is an exciting ti...	https://www.li...	ComplexOnsit...	Knowledge, Ski...	Part-time	32905	0	Palm ...	28631247	2264355

Hình 16: Bảng **Posting**.

- **Bảng Posting_State**:

```
def Posting_State():
    postings = pd.read_csv('postings.csv')
    Posting_State = postings[['original_listed_time', 'listed_time',
        'applies', 'views', 'expiry']].copy()
    Posting_State['original_listed_time'] = pd.to_datetime(
        Posting_State['original_listed_time'] / 1000, unit='s')
    Posting_State['expiry'] = pd.to_datetime(Posting_State['expiry']
        / 1000, unit='s')
    Posting_State['listed_time'] = pd.to_datetime(Posting_State['
        listed_time'] / 1000, unit='s')
    Posting_State['posting_id'] = range(1, len(Posting_State) + 1)
    Posting_State['applies'] = Posting_State['applies'].fillna(0).
        astype(int)
```

```
    Posting_State['views'] = Posting_State['views'].fillna(0).astype(
        int)
    Posting_State['apply_rate'] = Posting_State['applies'] /
        Posting_State['views'].replace(0, pd.NA)
    Posting_State['remaining_time'] = (Posting_State['expiry'] -
        Posting_State['original_listed_time']).dt.days
    column_order = ["posting_id", "original_listed_time", '
        listed_time', 'applies', 'views', 'expiry', 'apply_rate', '
        remaining_time']
    Posting_State = Posting_State[column_order]
    try:
        Posting_State.to_sql('Posting_State', con=engine, if_exists='
            append', index=False)
        print("Posting_State has been inserted successfully!")
    except Exception as e:
        print(f"Posting_State: An error occurred during insertion: {e
            }")
```

Listing 53: Các lệnh đọc và chèn dữ liệu vào bảng **Posting_State**.

	posting_state_id	timezone_offset	original_listed_time	listed_time	applies	views	expiry	posting_id	apply_rate	remaining_time
1	1	NULL	2024-04-17 23:4...	2024-04-17...	2	20	2024-05-17...	1	0.1	30
2	2	NULL	2024-04-11 17:5...	2024-04-11...	0	1	2024-05-11...	2	0	30
3	3	NULL	2024-04-16 14:2...	2024-04-16...	0	8	2024-05-16...	3	0	30
4	4	NULL	2024-04-12 04:2...	2024-04-12...	0	16	2024-05-12...	4	0	30
5	5	NULL	2024-04-18 14:5...	2024-04-18...	0	3	2024-05-18...	5	0	30
6	6	NULL	2024-04-18 16:0...	2024-04-18...	4	9	2024-05-18...	6	0.44444...	30
7	7	NULL	2024-04-11 18:4...	2024-04-11...	1	7	2024-05-11...	7	0.14285...	30
8	8	NULL	2024-04-06 22:4...	2024-04-06...	0	2	2024-10-03...	8	0	179
9	9	NULL	2024-04-05 20:2...	2024-04-05...	0	3	2024-05-05...	9	0	30
10	10	NULL	2024-04-07 02:1...	2024-04-07...	0	5	2024-05-07...	10	0	30

Hình 17: Bảng **Posting_State**.

- **Bảng Additional_Info**:

```
def Additional_Info():
    postings = pd.read_csv('postings.csv')
    Additional_Info = postings[['formatted_experience_level', '
        posting_domain', 'application_url', 'closed_time']].copy()
    Additional_Info['posting_id'] = range(1, len(Additional_Info) +
        1)
    Additional_Info['closed_time'] = pd.to_datetime(Additional_Info['
        closed_time'] / 1000, unit='s')
    column_order = ['formatted_experience_level', 'posting_domain', '
        application_url', 'closed_time', 'posting_id']
    Additional_Info = Additional_Info[column_order]
    Additional_Info.dropna(subset=['formatted_experience_level', '
        posting_domain', 'application_url', 'closed_time'], how='all'
        , inplace=True)
    try:
        Additional_Info.to_sql('Additional_Info', con=engine,
            if_exists='append', index=False)
        print("Additional_Info has been inserted successfully!")
    except Exception as e:
```

```
        print(f"Additional_Info: An error occurred during insertion:
            {e}")
```

Listing 54: Các lệnh đọc và chèn dữ liệu vào bảng **Additional_Info**.

	additional_info_id	formatted_experience_level	posting_domain	application_url	closed_time	posting_id
23	23	Associate	NULL	NULL	NULL	102
24	24	Entry level	jsv3.recruitics....	https://jsv3.r...	NULL	103
25	25	NULL	NULL	https://workf...	NULL	108
26	26	Entry level	NULL	NULL	NULL	110
27	27	NULL	NULL	NULL	2024-04-1...	111
28	28	NULL	NULL	https://jobs.g...	NULL	122
29	29	NULL	NULL	https://naven...	NULL	123
30	30	Mid-Senior level	NULL	NULL	NULL	130
31	31	NULL	NULL	https://emplo...	NULL	132
32	32	Mid-Senior level	NULL	https://crisp....	NULL	133

Hình 18: Bảng **Additional_Info**.

- **Bảng Salary**:

```
def Salary():
    salary = pd.read_csv('jobs/salaries.csv')
    posting = pd.read_csv('postings.csv')
    Salary_from_job_salary = salary[['salary_id', 'job_id', 'currency
        ', 'compensation_type', 'pay_period']].copy()
    Salary_from_posting = posting[['job_id', 'currency', '
        compensation_type', 'pay_period']].copy()
    Salary = pd.concat([Salary_from_job_salary, Salary_from_posting])
        .drop_duplicates(subset='job_id', keep='first')
    Salary['salary_id'] = range(1, len(Salary)+1)
    try:
        Salary.to_sql('Salary', con=engine, if_exists='append', index
            =False)
        print("Salary has been inserted successfully")
    except Exception as e:
        print(f'Salary: An error occurred during insertion: {e}')
```

Listing 55: Các lệnh đọc và chèn dữ liệu vào bảng **Salary**.

	salary_id	job_id	currency	compensation_type	pay_period
1	1	3884428798	USD	BASE_SALARY	HOURLY
2	2	3887470552	USD	BASE_SALARY	HOURLY
3	3	3884431523	USD	BASE_SALARY	YEARLY
4	4	3884911725	USD	BASE_SALARY	YEARLY
5	5	3887473220	USD	BASE_SALARY	HOURLY
6	6	3884432468	USD	BASE_SALARY	HOURLY
7	7	3884915161	USD	BASE_SALARY	YEARLY
8	8	3884428983	USD	BASE_SALARY	YEARLY
9	9	3884433248	USD	BASE_SALARY	HOURLY
10	10	3884429625	USD	BASE_SALARY	YEARLY

Hình 19: Bảng **Salary**.

- **Bảng Salary_Type**:

```
def Salary_Type():
    salary = pd.read_csv('jobs/salaries.csv')
    posting = pd.read_csv('postings.csv')
    Salary_from_job_salary = salary[['salary_id', 'job_id', '
        min_salary', 'max_salary', 'med_salary']].copy()
    Salary_from_posting = posting[['job_id', 'min_salary', '
        max_salary', 'med_salary']].copy()
    Salary = pd.concat([Salary_from_job_salary, Salary_from_posting])
        .drop_duplicates(subset='job_id', keep='first')
    Salary['salary_id'] = range(1, len(Salary)+1)
    Salary = Salary.rename(columns={
        'max_salary': 'max',
        'min_salary': 'min',
        'med_salary': 'med'
    })
    Salary_Type = pd.melt(Salary, id_vars=['salary_id'],
                          value_vars=['max', 'med', 'min'],
                          var_name='salary_type',
                          value_name='value')
    Salary_Type = Salary_Type.dropna(subset=['value'])
    try:
        Salary_Type.to_sql('Salary_Type', con=engine, if_exists='
            append', index=False)
        print("Salary_Type has been inserted successfully")
    except Exception as e:
        print(f'Salary_Type: An error occurred during insertion: {e}'
            )
```

Listing 56: Các lệnh đọc và chèn dữ liệu vào bảng **Salary_Type**.

	salary_type	salary_id	value
1	med	1	20
2	max	2	25
3	min	2	23
4	max	3	120000
5	min	3	100000
6	max	4	200000
7	min	4	10000
8	max	5	35
9	min	5	33
10	med	6	48.43

Hình 20: Bảng **Salary_Type**.

- **Bảng Requires**:

```
def Requires():
    skill = pd.read_csv('jobs/job_skills.csv')
    Requires = skill[['job_id', 'skill_abr']].copy()
    try:
```

```
        Requires.to_sql('Requires', con=engine, if_exists='append',
            index=False)
        print("Requires has been inserted successfully")
    except Exception as e:
        print(f'Requires: An error occurred during insertion: {e}')
```

Listing 57: Các lệnh đọc và chèn dữ liệu vào bảng **Requires**.

	skill_abr	job_id
1	MRKT	921716
2	SALE	921716
3	HCPR	1218575
4	HCPR	1829192
5	DSGN	2264355
6	ART	2264355
7	IT	2264355
8	MGMT	10998357
9	MNFC	10998357
10	IT	11009123

Hình 21: Bảng **Requires**.

- **Bảng Job_Has_Benefit**:

```
def Job_Has_Benefit():
    benefits = pd.read_csv('jobs/benefits.csv')
    Job_Has_Benefit = benefits[['job_id']].copy()
    Job_Has_Benefit['benefit_id'] = range(1, len(benefits)+1)
    try:
        Job_Has_Benefit.to_sql('Job_Has_Benefit', con=engine,
            if_exists='append', index=False)
        print("Job_Has_Benefit has been inserted successfully")
    except Exception as e:
        print(f'Job_Has_Benefit: An error occurred during insertion:
            {e}')
```

Listing 58: Các lệnh đọc và chèn dữ liệu vào bảng **Job_Has_Benefit**.

	job_id	benefit_id
1	23221523	14526
2	56482768	11999
3	56482768	12000
4	56482768	12001
5	69333422	59064
6	69333422	59065
7	69333422	59066
8	69333422	59067
9	95428182	8571
10	95428182	8572

Hình 22: Bảng **Job_Has_Benefit**.

- **Bảng Benefit_Has_Benefit_Type**:

```
def Benefit_Has_Benefit_Type():
    benefit = pd.read_csv('jobs/benefits.csv').copy()
    Job_Has_Benefit = benefit[['job_id', 'inferred']].drop_duplicates
        ().reset_index(drop=True)
    Job_Has_Benefit['benefit_id'] = range(1, len(Job_Has_Benefit) +
        1)
    benefit = pd.merge(benefit, Job_Has_Benefit[['job_id', '
        benefit_id']], on='job_id', how='left')
    Benefit_Type = benefit[['type']].drop_duplicates().reset_index(
        drop=True)
    Benefit_Type['benefit_type_id'] = range(1, len(Benefit_Type) + 1)
    benefit = pd.merge(benefit, Benefit_Type, on='type', how='left')
    Benefit_Has_Benefit_Type = benefit[['benefit_id', '
        benefit_type_id']].drop_duplicates().reset_index(drop=True)
    try:
        Benefit_Has_Benefit_Type.to_sql('Benefit_Has_Benefit_Type',
            con=engine, if_exists='append', index=False)
        print("Benefit_Has_Benefit_Type has been inserted
            successfully!")
    except Exception as e:
        print(f"Benefit_Has_Benefit_Type: An error occurred during
            insertion: {e}")
```

Listing 59: Các lệnh đọc và chèn dữ liệu vào bảng **Benefit_Has_Benefit_Type**.

	benefit_id	benefit_type_id
1	1	1
2	1	2
3	1	3
4	1	4
5	1	5
6	1	6
7	2	1
8	2	2
9	2	3
10	2	4

Hình 23: Bảng **Benefit_Has_Benefit_Type**.

- **Bảng Job_Has_Industry**:

```
def Job_Has_Industry():
    job_industries = pd.read_csv('jobs/job_industries.csv')
    Job_Has_Industry = job_industries[['job_id', 'industry_id']].
        dropna()
    try:
```

```
        Job_Has_Industry.to_sql('Job_Has_Industry', con=engine,
            if_exists='append', index=False)
        print("Job_Has_Industry has been inserted successfully!")
    except Exception as e:
        print(f"Job_Has_Industry: An error occurred during insertion:
            {e}")
```

Listing 60: Các lệnh đọc và chèn dữ liệu vào bảng **Job_Has_Industry**.

	job_id	industry_id
1	921716	44
2	1218575	14
3	2264355	89
4	9615617	142
5	10998357	32
6	11009123	50
7	23221523	9
8	35982263	122
9	56482768	100
10	56924323	51

Hình 24: Bảng **Job_Has_Industry**.

- **Bảng Company_Has_Industry**:

```
def Company_Has_Industry():
    company_industries = pd.read_csv('companies/company_industries.
        csv').copy()
    company_industries.rename(columns={
        'industry': 'industry_name'
    }, inplace=True)
    industries = pd.read_csv('mappings/industries.csv').copy()
    merged_df = pd.merge(company_industries, industries, on='
        industry_name', how='left')
    Company_Has_Industry = merged_df[['company_id', 'industry_id']].
        dropna()
    try:
        Company_Has_Industry.to_sql('Company_Has_Industry', con=
            engine, if_exists='append', index=False)
        print("Company_Has_Industry has been inserted successfully!")
    except Exception as e:
        print(f"Company_Has_Industry: An error occurred during
            insertion: {e}")
```

Listing 61: Các lệnh đọc và chèn dữ liệu vào bảng **Company_Has_Industry**.

	company_id	industry_id
1	1009	96
2	1016	14
3	1025	96
4	1028	96
5	1033	11
6	1035	4
7	1038	11
8	1043	147
9	1044	96
10	1052	8

Hình 25: Bảng **Company_Has_Industry**.

- **Bảng Company_Speciality**:

```
def Company_Speciality():
    specialities = pd.read_csv('companies/company_specialities.csv')
    specialities.dropna(inplace=True)
    max_length = 255
    specialities['speciality'] = specialities['speciality'].apply(
        lambda x: x[:max_length] if len(x) > max_length else x)
    specialities = specialities[['speciality']]
    try:
        specialities.to_sql('Company_Speciality', con=engine,
            if_exists='append', index=False)
        print("Company_Speciality has been inserted successfully!")
    except Exception as e:
        print(f"Company_Speciality: An error occurred during
            insertion: {e}")
```

Listing 62: Các lệnh đọc và chèn dữ liệu vào bảng **Company_Speciality**.

	speciality_id	speciality
1	1	window replacement
2	2	patio door replacement
3	3	Commercial Banking
4	4	Retail Banking
5	5	Mortgage
6	6	Private Banking
7	7	Trust Services
8	8	Insurance
9	9	Wealth Management
10	10	Advertising

Hình 26: Bảng **Company_Speciality**.

- **Bảng Company_Has_Speciality**:

```
def Company_Has_Speciality():
    company_specialities = pd.read_csv('companies/
        company_specialities.csv')
    Company_Has_Speciality = company_specialities[['company_id']].
        copy()
    Company_Has_Speciality['speciality_id'] = range(1, len(
        company_specialities) + 1)
    try:
        Company_Has_Speciality.to_sql('Company_Has_Speciality', con=
            engine, if_exists='append', index=False)
        print("Company_Has_Speciality has been inserted successfully!
            ")
    except Exception as e:
        print(f'Company_Has_Speciality: An error occurred during
            insertion: {e}')
```

Listing 63: Các lệnh đọc và chèn dữ liệu vào bảng **Company_Has_Speciality**.

	company_id	speciality_id
1	1009	18881
2	1009	18882
3	1009	18883
4	1009	18884
5	1009	18885
6	1009	18886
7	1009	18887
8	1009	18888
9	1009	18889
10	1009	18890

Hình 27: Bảng **Company_Has_Speciality**.

- **Bảng Employee_Count**:

```
def Employee_Count():
    employee_counts = pd.read_csv('companies/employee_counts.csv')
    employee_counts['time_recorded'] = employee_counts['time_recorded
        '].apply(safe_convert_timestamp)
    employee_counts.dropna(inplace=True)
    employee_counts = employee_counts.drop_duplicates(subset=['
        company_id', 'time_recorded'])
    try:
        employee_counts[['company_id', 'employee_count', '
            follower_count', 'time_recorded']].to_sql('Employee_Count
            ', con=engine, if_exists='append', index=False)
        print("Employee_Count has been inserted successfully!")
    except Exception as e:
        print(f"Error checking or inserting data: {e}")
```

Listing 64: Các lệnh đọc và chèn dữ liệu vào bảng **Employee_Count**.

	time_recorded	company_id	employee_count	follower_count
1	2024-04-05 19:42:53...	5887	1426	975931
2	2024-04-05 19:42:53...	11056	14122	2371805
3	2024-04-05 19:42:53...	12828	2659	43975
4	2024-04-05 19:42:53...	20300	1053	6554
5	2024-04-05 19:42:53...	23656	622	266464
6	2024-04-05 19:42:53...	33218	191	36335
7	2024-04-05 19:42:53...	391906	186	32508
8	2024-04-05 19:42:53...	729238	268	41433
9	2024-04-05 19:42:53...	766849	24	68011
10	2024-04-05 19:42:53...	878353	52	26397

Hình 28: Bảng **Employee_Count**.

- **Bảng Company_Location**:

```
def split_address(address):
    if not isinstance(address, str):
        return None, None
    match = re.match(r'^\s*(\d+)\s+(.+)$|^(.+?)\s+(\d+)\s*$', address
        )
    if match:
        if match.group(1):
            return match.group(1), match.group(2)
        else:
            return match.group(4), match.group(3)
    return None, address

def Company_Location():
    company = pd.read_csv('companies/companies.csv')
    Company_Location = company[['zip_code', 'address', 'state', '
        country', 'city', 'company_id']].copy()
    Company_Location.rename(columns={
        'zip_code': 'zipcode'
    }, inplace=True)
    Company_Location = Company_Location[Company_Location['zipcode'].
        apply(lambda x: len(x) <= 15 if isinstance(x, str) else True)
        ]
    Company_Location[['number', 'street']] = Company_Location['
        address'].apply(split_address).apply(pd.Series)
    Company_Location.replace(['0', 0, '-'], np.nan, inplace=True)
    Company_Location.dropna(subset=['zipcode', 'number', 'street', '
        state', 'country', 'city'], how='all', inplace=True)
    Company_Location = Company_Location[['zipcode', 'number', 'street
        ',
                                         'state', 'country', 'city',
                                            'company_id']]
    try:
        Company_Location.to_sql('Company_Location', con=engine,
```

```
            if_exists='append', index=False)
        print("Company_Location has been inserted successfully!")
    except Exception as e:
        print(f"Company_Location: An error occurred during insertion:
            {e}")
```

Listing 65: Các lệnh đọc và chèn dữ liệu vào bảng **Company_Location**.

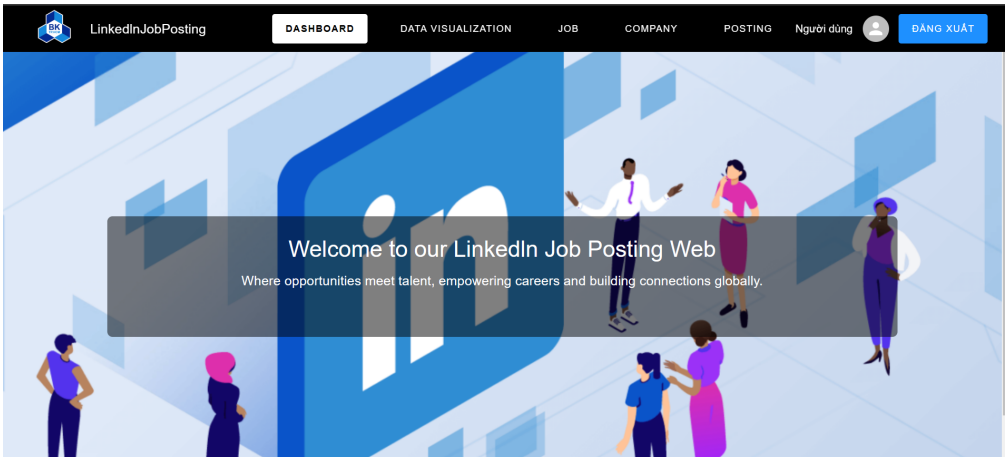
	location_id	zipcode	number	street	state	country	city	company_id
1	1	10504	NULL	International Business Machines Corp.	NY	US	Armonk, New York	1009
2	2	NULL	NULL	NULL	NULL	US	Chicago	1016
3	3	77389	1701	E Mossy Oaks Rd Spring	Texas	US	Houston	1025
4	4	78741	2300	Oracle Way	Texas	US	Austin	1028
5	5	NULL	NULL	Grand Canal Harbour	NULL	IE	Dublin 2	1033
6	6	98052	1	Microsoft Way	Washington	US	Redmond	1035
7	7	NULL	NULL	Worldwide	NULL	OO	Worldwide	1038
8	8	80333	1	Werner-von-Siemens-Straße	NULL	DE	Munich	1043
9	9	NULL	1	Embankment Place	NULL	GB	NULL	1044
10	10	75202	208	S. Akard Street	TX	US	Dallas	1052

Hình 29: Bảng **Company_Location**.

5 Xây dựng trang Content Management System

Xây dựng trang Content Management System demo đọc dữ liệu, lưu trữ, xoá, cập nhật dành cho người quản trị hệ thống (Lưu lại thời gian đáp ứng/ xử lý dữ liệu của các API). Ý tưởng ban đầu được xây dựng bằng công nghệ Figma: https://bit.ly/Figma-CO3127-L05-N02.

1. **Giao diện Trang chủ (Home) và Thanh điều hướng (Navigation Bar):**



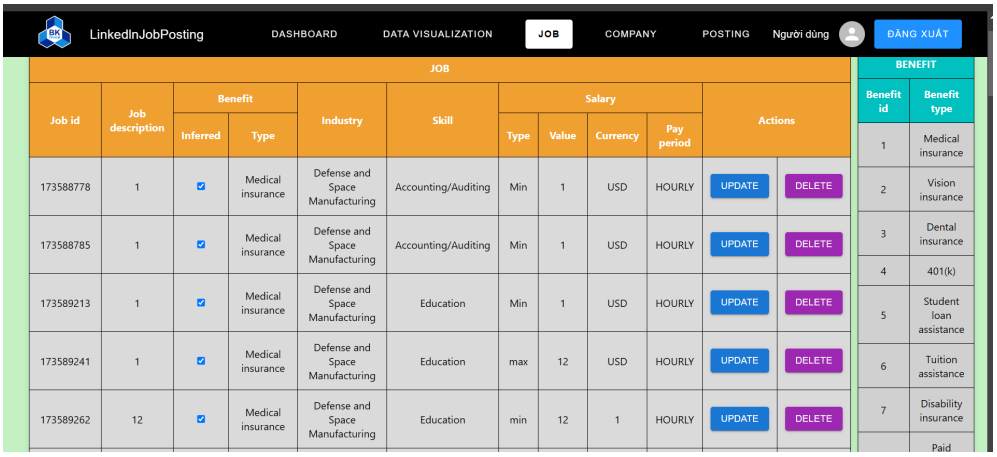
Hình 30: Giao diện **Trang chủ (Home)** và **Thanh điều hướng (Navigation Bar)**.

- Thanh Điều Hướng (Navigation Bar) bao gồm các lựa chọn:

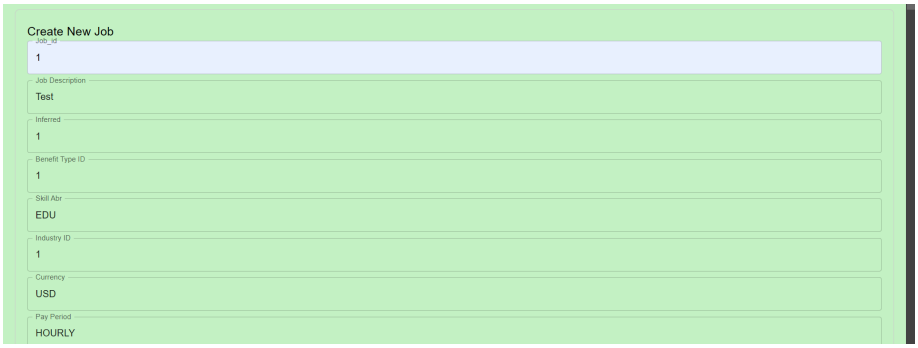
- ***Dashboard:*** dẫn đến giao diện biểu đồ 1 và ***Bánh răng:*** chọn đường dẫn trực tiếp đến biểu đổ mong muốn.

- ***Job, Company, Posting:*** dẫn đến giao diện chứa dữ liệu liên quan đến lần lượt là Job, Company và Posting.

2. **Giao diện chứa dữ liệu liên quan đến Job:**



Hình 31: Giao diện chứa dữ liệu liên quan đến **Job**.



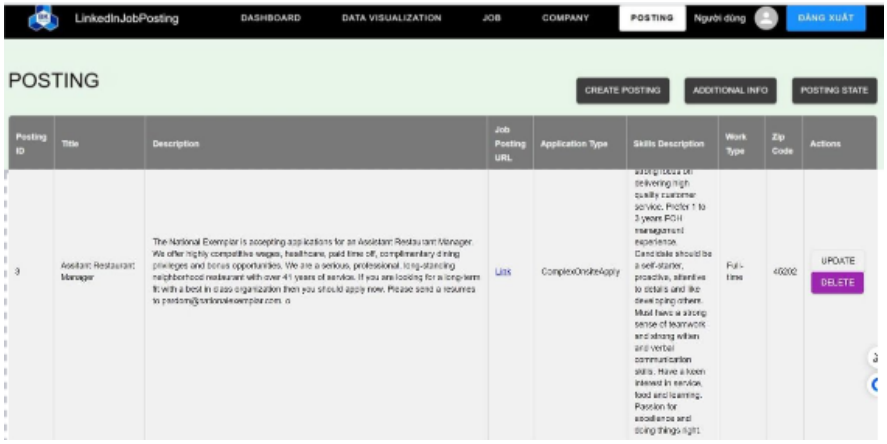
Hình 32: Giao diện thêm **(CREATE)** dữ liệu liên quan đến **Job**.

Hình 33: Giao diện cập nhật **(UPDATE)** dữ liệu liên quan đến **Job**.

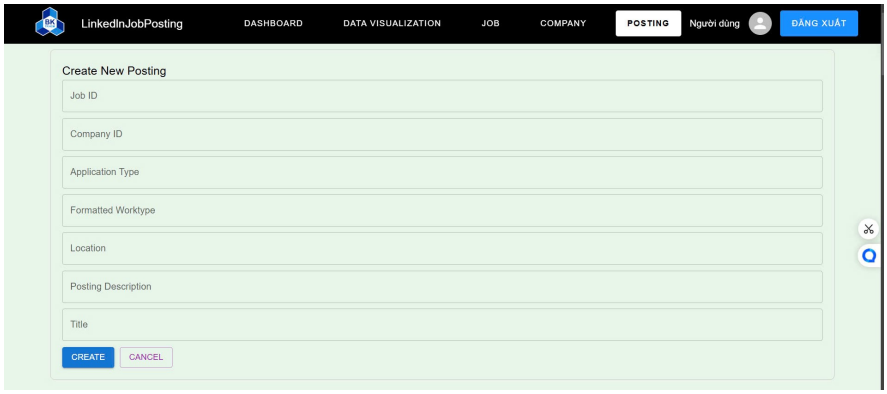
Hình 34: Giao diện xóa **(DELETE)** dữ liệu liên quan đến **Job**.

Giao diện **Job** ngoài việc có thể xem chi tiết các thông tin chi tiết về job ta còn có thể xem các thông tin về các phúc lợi, mức lương. Đồng thời, ta có thể thực hiện thêm, xóa, sửa các công việc trong LinkedIn.

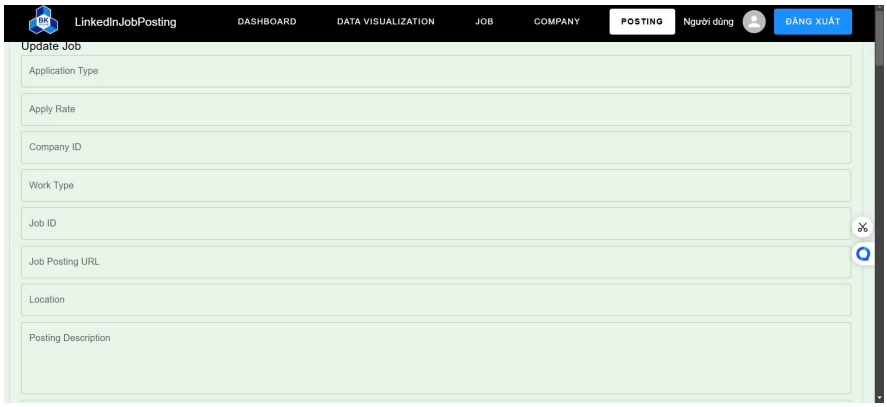
3. **Giao diện chứa dữ liệu liên quan đến Posting:**



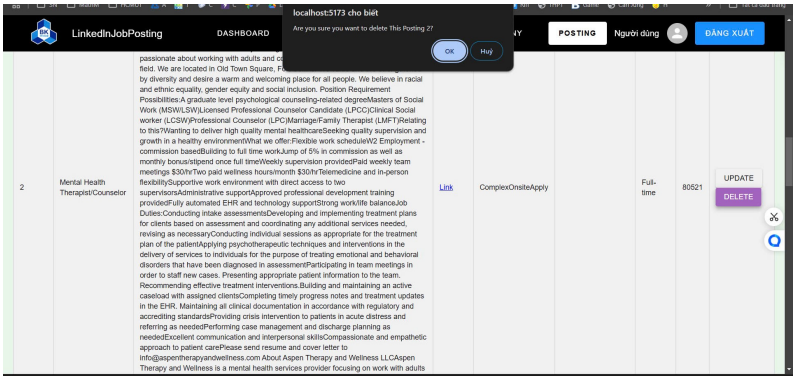
Hình 35: Giao diện chứa dữ liệu liên quan đến **Posting**.



Hình 36: Giao diện thêm **(CREATE)** dữ liệu liên quan đến **Posting**.



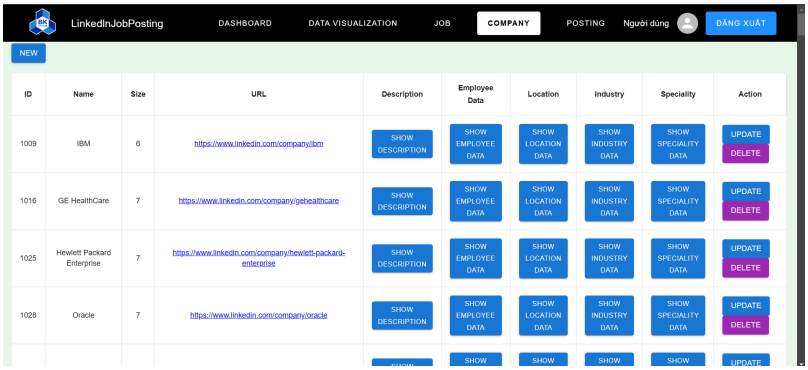
Hình 37: Giao diện cập nhật **(UPDATE)** dữ liệu liên quan đến **Posting**.



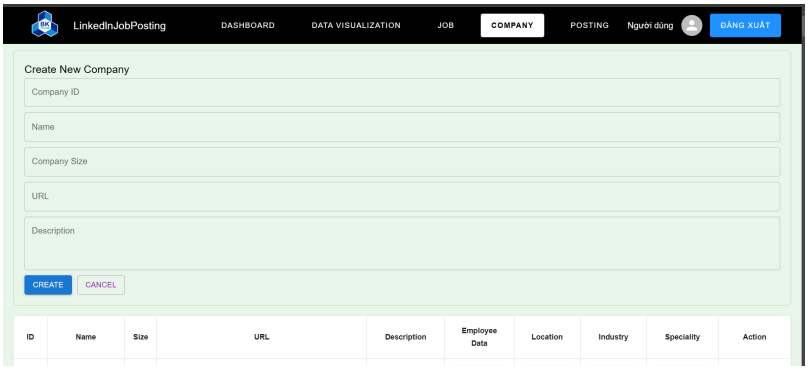
Hình 38: Giao diện xóa **(DELETE)** dữ liệu liên quan đến **Posting**.

Giao diện **Posting** hỗ trợ xem thông tin cơ bản của các bài đăng, trạng thái và thông tin bổ sung như yêu cầu kinh nghiệm, link công việc, tên và mô tả công việc. Ngoài ra, còn có các chức năng như thêm, cập nhật và xóa bài đăng để quản lý Posting.

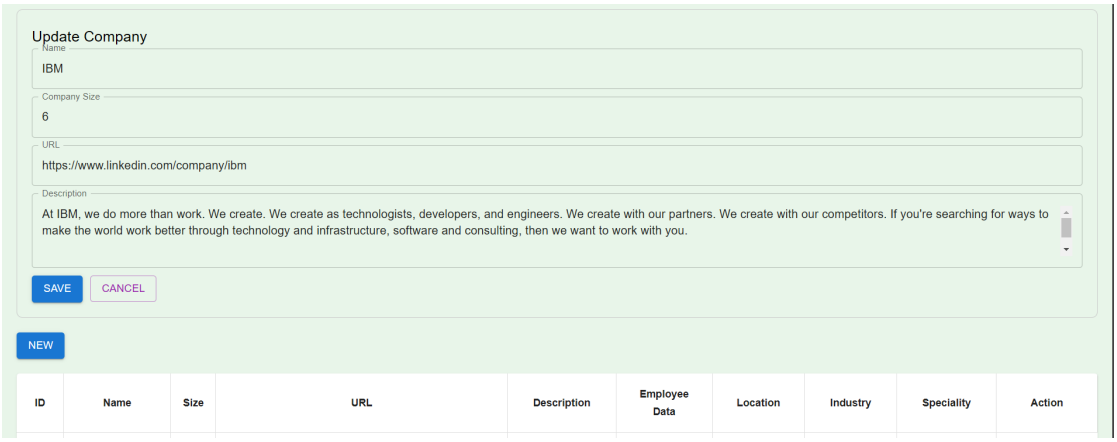
4. **Giao diện chứa dữ liệu liên quan đến Company:**



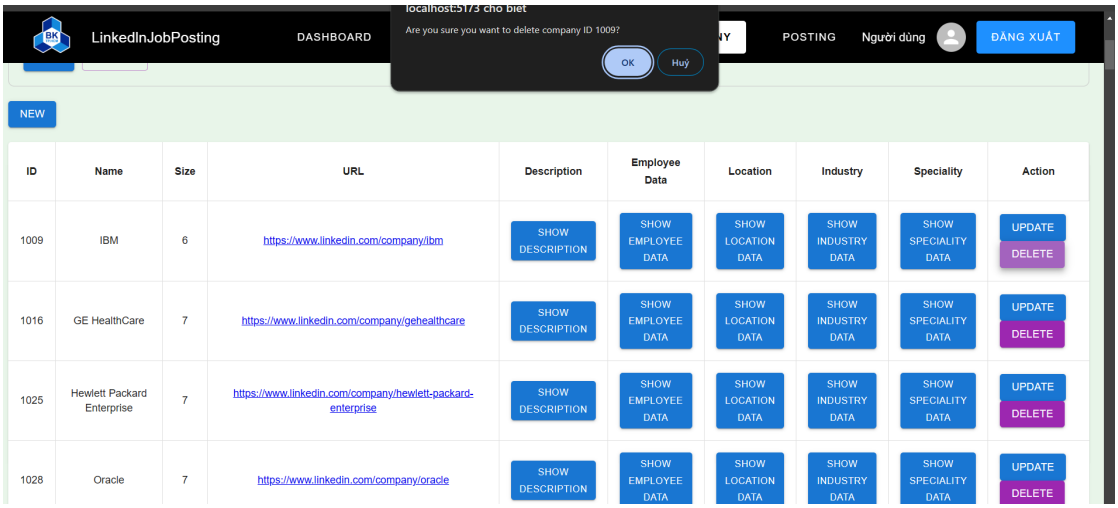
Hình 39: Giao diện chứa dữ liệu liên quan đến **Company**.



Hình 40: Giao diện thêm **(CREATE)** dữ liệu liên quan đến **Company**.



Hình 41: Giao diện cập nhật **(UPDATE)** dữ liệu liên quan đến **Company**.



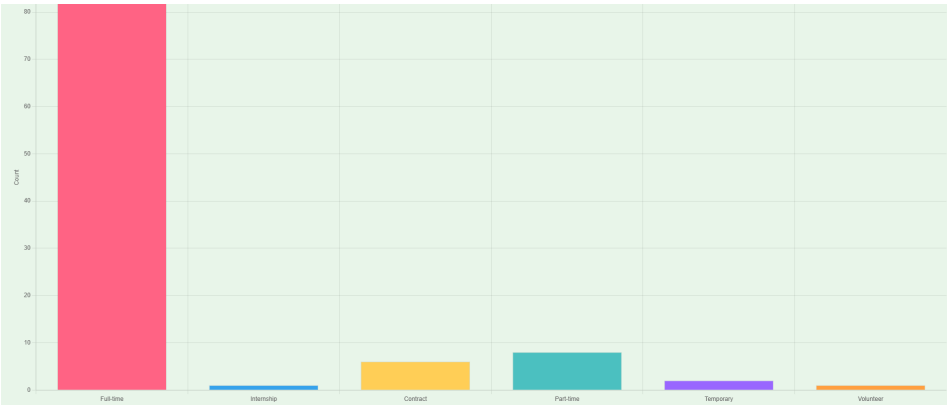
Hình 42: Giao diện xóa **(DELETE)** dữ liệu liên quan đến **Company**.

Giao diện **Company** ngoài việc có thể xem chi tiết các thông tin chi tiết về company ta còn có thể xem các thông tin về nhân sự, vị trí, ngành và thế mạnh của công ty. Đồng thời, ta có thể thực hiện thêm, xóa, sửa thông tin của công ty trong LinkedIn.

6 Xây dựng giao diện Dashboard hiển thị các biểu đồ

Xây dựng giao diện Dashboard hiển thị các biểu đồ (Lưu lại thời gian đáp ứng/ xử lý dữ liệu của các API và các câu truy vấn liên quan). Ý tưởng ban đầu được xây dựng bằng công nghệ Figma: https://bit.ly/Figma-CO3127-L05-N02.

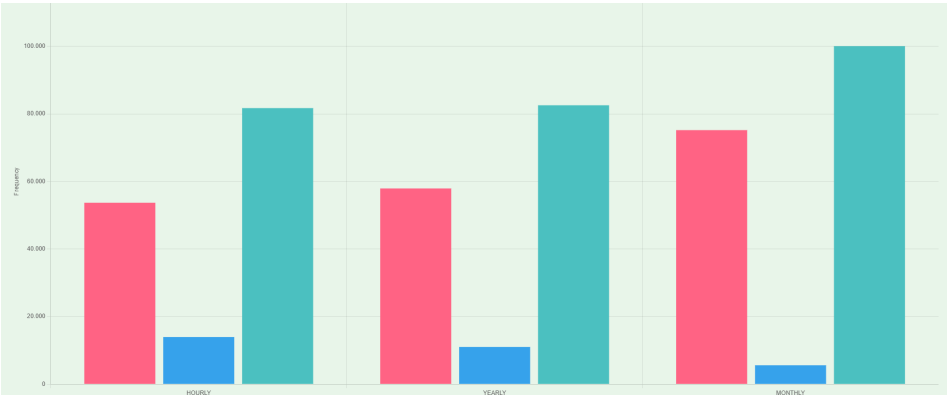
1. **Giao diện Dashboard 1. Worktype Distribution:**



Hình 43: Giao diện **Dashboard 1. Worktype Distribution**.

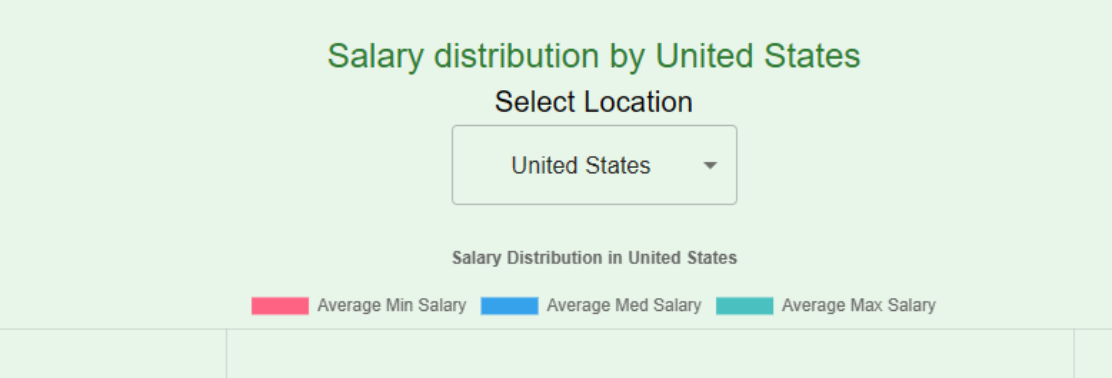
Biểu đồ **Dashboard 1. Worktype Distribution** thể hiện tỷ lệ các loại hình công việc bao gồm toàn thời gian, bán thời gian, hợp đồng và các loại khác. Mỗi loại hình công việc được phân biệt bằng màu sắc riêng biệt, giúp dễ dàng so sánh và xác định xu hướng phổ biến trên thị trường lao động hiện tại.

2. **Giao diện Dashboard 2. Salary Distribution by Pay Period:**



Hình 44: Giao diện **Dashboard 2. Salary Distribution by Pay Period**.

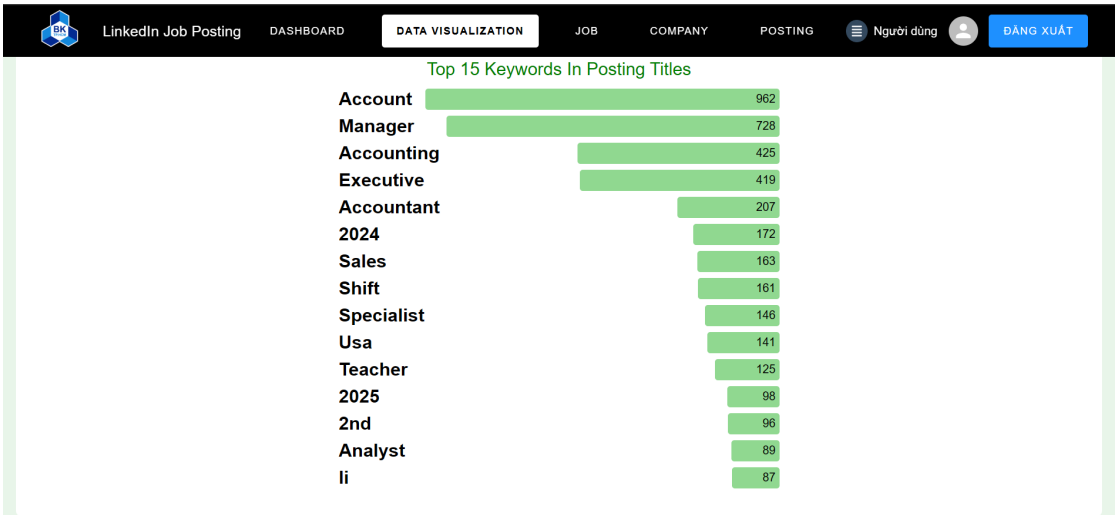
Biểu đồ **Dashboard 2. Salary Distribution by Pay Period** minh họa mức lương trung bình được trả theo các kỳ lương như lương giờ, lương ngày, và lương tháng. Các mức lương này được so sánh với mức trung bình tại New York, với các điểm dữ liệu được thể hiện bằng các thanh màu tương ứng từng nhóm kỳ lương.



Hình 45: Giao diện **Filter Dashboard 2. Salary Distribution by Pay Period**.

Filter Dashboard 2. Salary Distribution by Pay Period thể hiện sự phân bố mức lương theo vị trí địa lý, với từng khu vực được đánh dấu bằng màu sắc riêng biệt. Biểu đồ áp dụng các bộ lọc cụ thể để tập trung vào những khu vực tuyển dụng nhiều nhất.

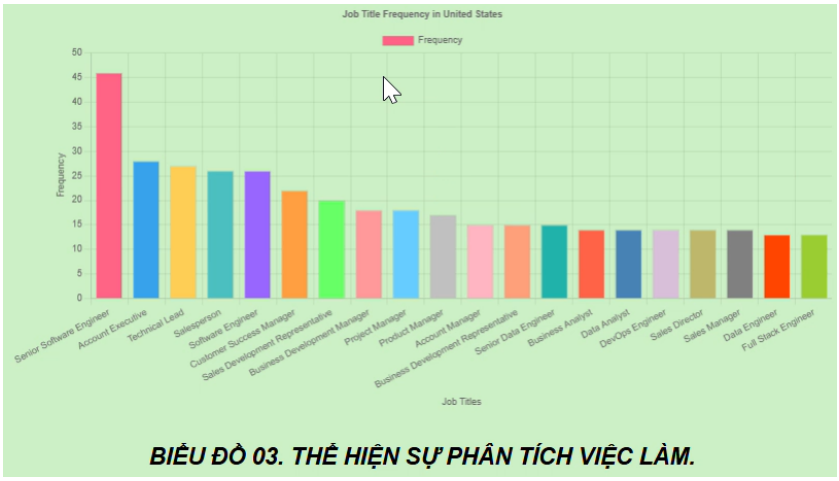
3. **Giao diện Dashboard 3. Posting Title:**



Hình 46: Giao diện **Dashboard 3. Posting Title**.

Biểu đồ **Dashboard 3. Posting Title** hiển thị 15 từ khóa phổ biến nhất trong tiêu đề các bài đăng tuyển dụng. Các từ khóa được xếp hạng theo số lần xuất hiện, với các cột biểu diễn chiều cao tỷ lệ thuận với tần suất, giúp nhanh chóng nhận biết các từ khóa quan trọng.

4. **Giao diện Dashboard 4. Job Title:**



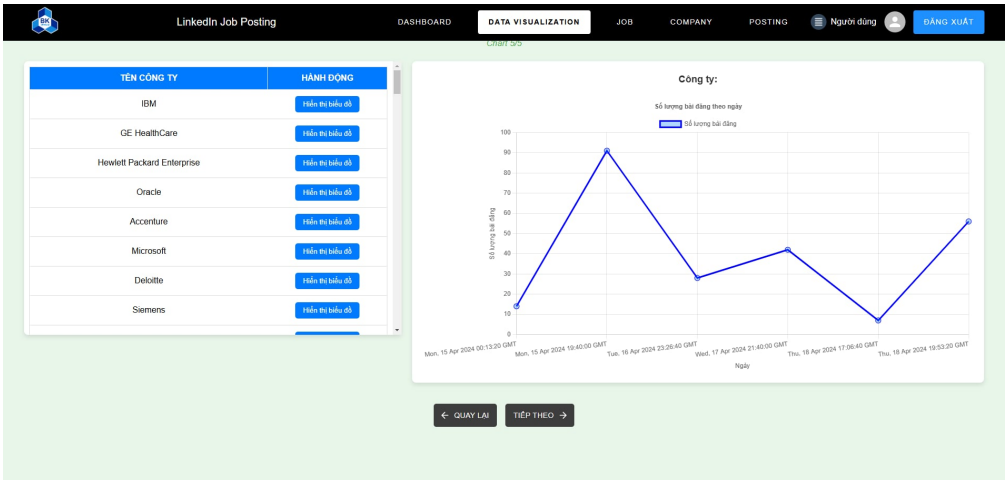
Hình 47: Giao diện **Dashboard 4. Job Title**.

Biểu đồ **Dashboard 4. Job Title** trình bày tần suất xuất hiện của các chức danh công việc phổ biến nhất. Các thanh biểu đồ được nhóm lại theo khu vực địa lý, làm nổi bật sự khác biệt về nhu cầu tuyển dụng giữa các khu vực.

Hình 48: Giao diện **Filter Dashboard 4. Job Title Filter By Location**.

Filter Dashboard 4. Job Title Filter By Location thể hiện sự phân bố công việc theo vị trí địa lý, với từng khu vực được đánh dấu bằng màu sắc riêng biệt. Biểu đồ áp dụng các bộ lọc cụ thể để tập trung vào những khu vực tuyển dụng nhiều nhất.

5. **Giao diện Dashboard 5. Postings Count:**



Hình 49: Giao diện **Dashboard 5. Postings Count**.

Biểu đồ **Dashboard 5. Postings Count** cho biết số lượng bài đăng tuyển dụng từ các công ty. Các cột biểu đồ được sắp xếp theo thứ tự giảm dần, giúp dễ dàng nhận biết những công ty có hoạt động tuyển dụng sôi nổi nhất.

7 Ứng dụng kết nối cơ sở dữ liệu

7.1 Tổng quan công nghệ

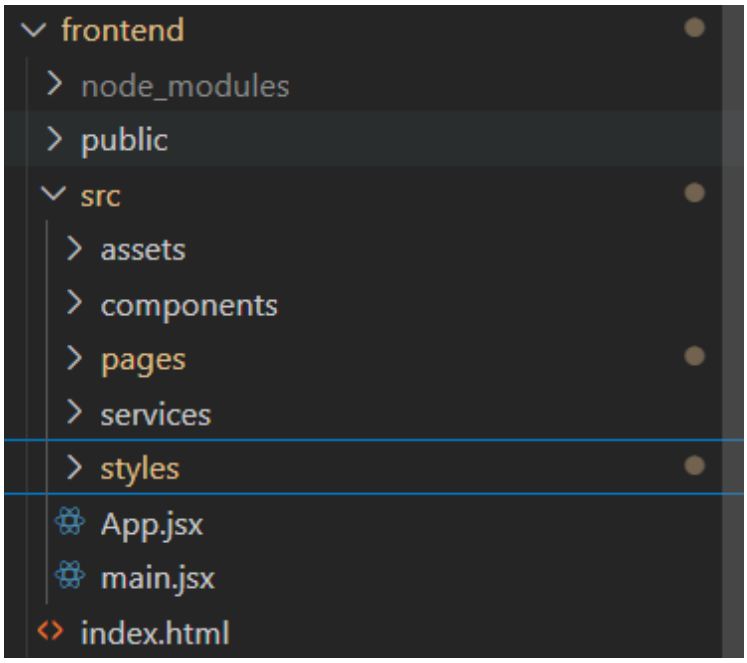
7.1.1 Công nghệ sử dụng

1. ***Môi trường phát triển:*** Website application.
2. ***Ngôn ngữ lập trình:***
   - *Frontend:* Javascript (thư viện ReactJs).
   - *Backend:* Javascipt, Python.
3. ***Hệ cơ sở dữ liệu:*** SQL Server.

7.1.2 Tổ chức thư mục code

7.1.2.a Tổ chức thư mục FrontEnd

Thư mục **FrontEnd** chứa mã nguồn giao diện người dùng, được xây dựng bằng **REACT**.



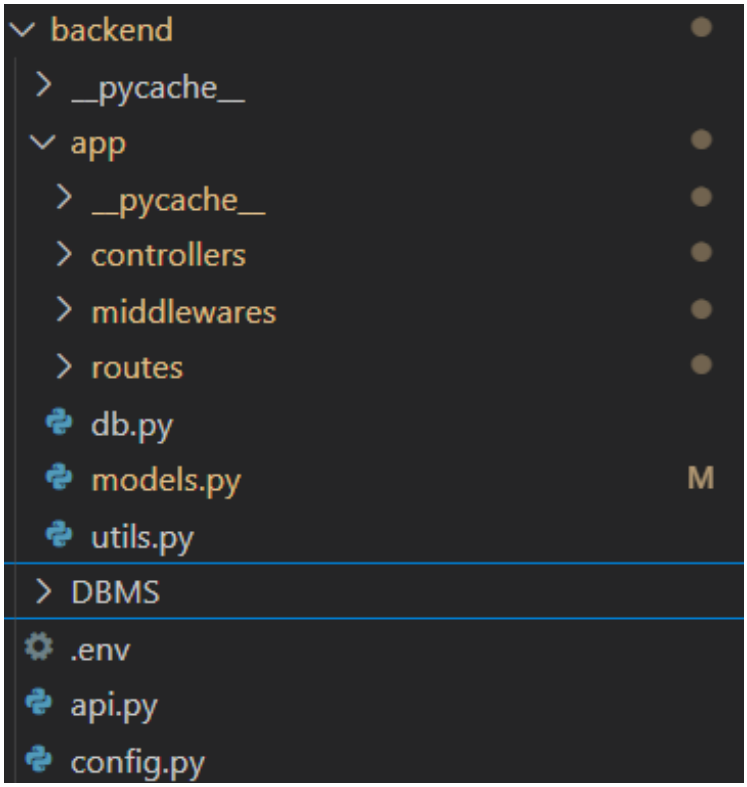
Hình 50: Giao diện **Tổ chức thư mục frontend**.

- ***Components:*** Thư mục `components/` lưu trữ các thành phần giao diện tái sử dụng, ví dụ: `Header.jsx` và `Footer.jsx`.
- ***Pages:*** Thư mục `pages/` chứa các trang chính của ứng dụng, mỗi trang tương ứng với một route. Ví dụ: `Home.jsx` là trang chủ.
- ***Services:*** Thư mục `services/` quản lý các hàm gọi API, sử dụng `axios` để giao tiếp với backend. Ví dụ: `api.js` xử lý việc lấy dữ liệu từ API.

- ***Styles:*** Thư mục `styles/` chứa các tệp CSS hoặc SCSS để định nghĩa kiểu dáng của ứng dụng.
- ***App.jsx:*** Tệp chính nơi tất cả các thành phần và route được ghép lại với nhau để tạo nên ứng dụng.
- ***main.jsx:*** Entry point của ứng dụng, nơi khởi chạy React.

7.1.2.b Tổ chức thư mục BackEnd

Thư mục **BackEnd** chứa toàn bộ logic phía Server.



Hình 51: Giao diện **Tổ chức thư mục Backend**.

- ***Controllers:*** Thư mục `controllers/` chứa các tệp xử lý logic nghiệp vụ (business logic). Ví dụ: `company_controller.py` xử lý các yêu cầu liên quan đến công ty.
- ***Middlewares:*** Thư mục `middlewares/` chứa các logic xử lý trung gian, như xác thực yêu cầu (`company_middleware.py`)
- ***Routes:*** Thư mục `routes/` định nghĩa các API endpoint, ánh xạ URL đến các hàm trong `controllers/`.
- ***Models:*** Thư mục `models/` chứa các mô hình cơ sở dữ liệu, được xây dựng bằng ORM như SQLAlchemy.

- `config.py`: Quản lý cấu hình của ứng dụng như kết nối cơ sở dữ liệu, khóa bí mật, v.v.
- `api.py`: Điểm khởi chạy ứng dụng BackEnd (entry point) để khởi tạo Server (Flask/ FastAPI).
- `.env`: Chứa các biến môi trường nhạy cảm như URL của cơ sở dữ liệu hoặc API keys.

7.1.2.c Luồng xử lý phía BackEnd

Khi một yêu cầu (request) được gửi đến BackEnd, luồng xử lý diễn ra như sau:

1. URL yêu cầu được ánh xạ tới một route trong `routes/`.
2. Route gọi một hàm trong `controllers/` để xử lý logic nghiệp vụ.
3. Nếu cần xác thực hoặc kiểm tra quyền, yêu cầu sẽ đi qua middleware trong `middlewares/`.
4. Controller tương tác với cơ sở dữ liệu thông qua các mô hình (models) trong `models/`.
5. Kết quả được xử lý và trả về cho client dưới dạng JSON.

7.1.3 Khai báo kết nối với DBMS

```
import os

class Config:
    SQLALCHEMY_DATABASE_URI = os.getenv(
        "DATABASE_URI",
        "mssql+pyodbc://ZRMR/KTDL?driver=ODBC+Driver+17+for+SQL+Server"
    )
    SQLALCHEMY_TRACK_MODIFICATIONS = False
```

Listing 66: Khai báo kết nối với DBMS.

7.2 Hiện thực

7.2.1 Job

1. **Các API của Job**

```
http ://127.0.0.1:5000/job
http ://127.0.0.1:5000/add_job
http ://127.0.0.1:5000/update_job/<job_id>
http ://127.0.0.1:5000/delete_job/<job_id>
```

Listing 67: Các API của **Job**.

2. **Hàm lấy danh sách các công việc Job từ cơ sở dữ liệu**

```
@GN_bp.route('/job', methods=['GET'])
def job():
    try:
        with engine.connect() as connection:
            result = connection.execute(text("""
```

```
                SELECT TOP 100
                    A.job_id,
                    A.job_description,
                    C.inferred,
                    D.benefit_type_id,
                    L.type,
                    H.industry_id,
                    H.industry_name,
                    F.skill_abr,
                    F.skill_name,
                    K.salary_type,
                    K.value,
                    J.currency,
                    J.pay_period
                FROM Job A
                JOIN Job_Has_Benefit B ON A.job_id = B.job_id
                JOIN Benefit C ON B.benefit_id = C.benefit_id
                JOIN Benefit_Has_Benefit_Type D ON C.benefit_id = D.
                    benefit_id
                JOIN Benefit_Type L on D.benefit_type_id = L.
                    benefit_type_id
                JOIN Requires E ON A.job_id = E.job_id
                JOIN Skill F ON E.skill_abr = F.skill_abr
                JOIN Job_Has_Industry G ON A.job_id = G.job_id
                JOIN Company_Industry H ON G.industry_id = H.
                    industry_id
                JOIN Salary J ON A.job_id = J.job_id
                JOIN Salary_Type K ON J.salary_id = K.salary_id;
            """))
            rows = [dict(row._mapping) for row in result]

            return jsonify(
                {
                    "success": True,
                    "data": rows
                }
            )

    except Exception as e:
        return jsonify(
            {
                "success": False,
                "message": str(e)
            }
        )
```

Listing 68: Hàm lấy danh sách các công việc **Job** từ cơ sở dữ liệu.

3. **Hàm tạo công việc Job mới từ cơ sở dữ liệu**

```
@GN_bp.route('/add_job', methods=['POST'])
def add_job():
    try:
        job_data = request.get_json()
```

```
        job_id = job_data['job_id']
        job_description = job_data['job_description']
        benefit_id = generate_unique_id()
        inferred = job_data['inferred']
        benefit_type_id = job_data['benefit_type_id']
        skill_abr = job_data['skill_abr']
        industry_id = job_data['industry_id']
        currency = job_data['currency']
        pay_period = job_data['pay_period']
        salary_id = generate_unique_id()###
        salary_type = job_data['salary_type']
        value = job_data['value']
        with engine.connect() as connection:
            with connection.begin():
                connection.execute(text("""
                    INSERT INTO Job (job_id, job_description) VALUES
                        (:job_id, :job_description)
                """), {'job_id': job_id, 'job_description':
                    job_description})

                connection.execute(text("""
                    SET IDENTITY_INSERT Benefit ON
                    INSERT INTO Benefit (benefit_id, inferred) VALUES
                         (:benefit_id, :inferred)
                    SET IDENTITY_INSERT Benefit OFF
                """), {'benefit_id': benefit_id, 'inferred': inferred
                    })
                connection.execute(text("""
                    INSERT INTO Job_Has_Benefit (job_id, benefit_id)
                        VALUES (:job_id, :benefit_id)
                """), {'job_id': job_id, 'benefit_id': benefit_id})
                connection.execute(text("""
                    INSERT INTO Benefit_Has_Benefit_Type (benefit_id,
                         benefit_type_id) VALUES (:benefit_id, :
                        benefit_type_id)
                """), {'benefit_id': benefit_id, 'benefit_type_id':
                    benefit_type_id})
                connection.execute(text("""
                    INSERT INTO Requires (job_id, skill_abr) VALUES
                        (:job_id, :skill_abr)
                """), {'job_id': job_id, 'skill_abr': skill_abr})

                connection.execute(text("""
                    INSERT INTO Job_Has_Industry (job_id, industry_id
                        ) VALUES (:job_id, :industry_id)
                """), {'job_id': job_id, 'industry_id': industry_id})
                connection.execute(text("""
                    SET IDENTITY_INSERT Salary ON
                    INSERT INTO Salary (salary_id, job_id, currency,
                        pay_period) VALUES (:salary_id, :job_id, :
                        currency, :pay_period)
                    SET IDENTITY_INSERT Salary OFF
                """), {'salary_id': salary_id, 'job_id': job_id, '
```

```
                    currency': currency, 'pay_period': pay_period})
                connection.execute(text("""
                    INSERT INTO Salary_Type (salary_id, salary_type,
                        value) VALUES (:salary_id, :salary_type, :
                        value)
                """), {'salary_id': salary_id, 'salary_type':
                    salary_type, 'value': value})
        return jsonify({
            "success": True,
            "message": f"Job {job_id} created successfully"
        }), 201
    except Exception as e:
        return jsonify({
            "success": False,
            "message": str(e)
        }), 400
```

Listing 69: Hàm tạo công việc **Job** mới từ cơ sở dữ liệu.

4. **Hàm xóa công việc Job từ cơ sở dữ liệu**

```
@GN_bp.route('/delete_job/<job_id>', methods=['DELETE'])
def delete_job(job_id):
    try:
        with engine.connect() as connection:
            with connection.begin():
                connection.execute(text("""
                    DELETE FROM Job_Has_Benefit WHERE job_id = :
                        job_id
                """), {'job_id': job_id})
                connection.execute(text("""
                    DELETE FROM Benefit_Has_Benefit_Type WHERE
                        benefit_id IN (
                        SELECT benefit_id FROM Job_Has_Benefit WHERE
                            job_id = :job_id
                    )
                """), {'job_id': job_id})
                connection.execute(text("""
                    DELETE FROM Requires WHERE job_id = :job_id
                """), {'job_id': job_id})
                connection.execute(text("""
                    DELETE FROM Job_Has_Industry WHERE job_id = :
                        job_id
                """), {'job_id': job_id})
                connection.execute(text("""
                    DELETE FROM Salary_Type WHERE salary_id IN (
                        SELECT salary_id FROM Salary WHERE job_id = :
                            job_id
                    )
                """), {'job_id': job_id})
                connection.execute(text("""
                    DELETE FROM Salary WHERE job_id = :job_id
                """), {'job_id': job_id})
                connection.execute(text("""
```

```
                    DELETE FROM Job WHERE job_id = :job_id
                """), {'job_id': job_id})
        return jsonify({
            "success": True,
            "message": f"Job with job_id {job_id} deleted
                successfully"
        }), 200
    except Exception as e:
        return jsonify({
            "success": False,
            "message": str(e)
        }), 400
```

Listing 70: Hàm xóa công việc **Job** từ cơ sở dữ liệu.

5. **Hàm cập nhật công việc Job từ cơ sở dữ liệu**

```
@GN_bp.route('/update_job/<job_id>', methods=['PUT'])
def update_job(job_id):
    try:
        update_data = request.get_json()
        with engine.connect() as connection:
            with connection.begin():
                if "job_description" in update_data:
                    connection.execute(text("""
                        UPDATE Job SET job_description = :
                            job_description WHERE job_id = :job_id
                    """), {'job_description': update_data['
                        job_description'], 'job_id': job_id})
                if "industry_id" in update_data:
                    connection.execute(text("""
                        UPDATE Job_Has_Industry
                        SET industry_id = :industry_id
                        WHERE job_id = :job_id
                    """), {'industry_id': update_data['industry_id'],
                         'job_id': job_id})
                if "currency" in update_data or "pay_period" in
                    update_data:
                    params = {'job_id': job_id}
                    set_clauses = []
                    if "currency" in update_data:
                        set_clauses.append("currency = :currency")
                        params['currency'] = update_data['currency']
                    if "pay_period" in update_data:
                        set_clauses.append("pay_period = :pay_period"
                            )
                        params['pay_period'] = update_data['
                            pay_period']
                    set_clause = ", ".join(set_clauses)
                    connection.execute(text(f"""
                        UPDATE Salary SET {set_clause} WHERE job_id =
                             :job_id
                    """), params)
```

```
                if "salary_type" in update_data or "value" in
                    update_data:
                    params = {'job_id': job_id}
                    set_clauses = []
                    if "salary_type" in update_data:
                        set_clauses.append("salary_type = :
                            salary_type")
                        params['salary_type'] = update_data['
                            salary_type']
                    if "value" in update_data:
                        set_clauses.append("value = :value")
                        params['value'] = update_data['value']
                    set_clause = ", ".join(set_clauses)
                    connection.execute(text(f"""
                        UPDATE Salary_Type
                        SET {set_clause}
                        WHERE salary_id = (
                            SELECT salary_id FROM Salary WHERE job_id
                                = :job_id
                        )
                    """), params)

        return jsonify({
            "success": True,
            "message": f"Job with job_id {job_id} updated
                successfully"
        }), 200

    except Exception as e:
        return jsonify({
            "success": False,
            "message": str(e)
        }), 400
```

Listing 71: Hàm cập nhật công việc **Job** từ cơ sở dữ liệu.

7.2.2 Posting

1. **Các API của Posting**

```
http://127.0.0.1:5000/api/v1/posting
http://127.0.0.1:5000/api/v1/posting_state
http://127.0.0.1:5000/api/v1/additional_information
```

Listing 72: Các API của **Posting**.

2. **Thiết lập các công cụ và module**

```
from flask import Blueprint, jsonify, request
from app.models import db, Posting, PostingState, AdditionalInfo
import traceback
```

Listing 73: Thiết lập các công cụ và module.

3. **Tạo Blueprint tên bài đăng Posting để tổ chức các route.**

```
posting_bp = Blueprint('posting', __name__)
```

Listing 74: Tạo Blueprint tên bài đăng **Posting** để tổ chức các route trong Flask.

4. **Các hàm chuyển đổi dữ liệu liên quan đến bài đăng Posting từ các bảng cơ sở dữ liệu thành định dạng từ điển**

```
from flask_sqlalchemy import SQLAlchemy
import base64
db = SQLAlchemy()

class Salary(db.Model):
    __tablename__ = 'salary'
    salary_id = db.Column(db.BigInteger, primary_key=True)
    job_id = db.Column(db.BigInteger, nullable=True)
    currency = db.Column(db.String(10), nullable=True)
    compensation_type = db.Column(db.String(50), nullable=True)
    pay_period = db.Column(db.String(50), nullable=True)
    def to_dict(self):
        return {
            'salary_id': self.salary_id,
            'job_id': self.job_id,
            'currency': self.currency,
            'compensation_type': self.compensation_type,
            'pay_period': self.pay_period
        }

class Salary_Type(db.Model):
    __tablename__ = 'salary_type'
    salary_id = db.Column(db.BigInteger, primary_key=True)
    salary_type = db.Column(db.String(10), nullable=False)
    value = db.Column(db.Float, nullable=True)
    def to_dict(self):
        return {
            'salary_id': self.salary_id,
            'salary_type': self.salary_type,
            'value': self.value,
        }

class Company(db.Model):
    __tablename__ = 'company'
    company_id = db.Column(db.BigInteger, primary_key=True)
    company_name = db.Column(db.String(255), nullable=True)
    company_size = db.Column(db.Integer, nullable=True)
    company_url = db.Column(db.String(255), nullable=True)
    description = db.Column(db.Text, nullable=True)
    postings = db.relationship('Posting', backref='company', lazy=
        True)
    def to_dict(self):
        return {
            'company_id': self.company_id,
            'company_name': self.company_name,
```

```
            'company_size': self.company_size,
            'company_url': self.company_url,
            'description': self.description
        }

class Job(db.Model):
    __tablename__ = 'job'
    job_id = db.Column(db.BigInteger, primary_key=True)
    job_description = db.Column(db.Text, nullable=True)
    skills = db.Column(db.String(max), nullable=True)
    postings = db.relationship('Posting', backref='job', lazy=True)
    def to_dict(self):
        return {
            'job_id': self.job_id,
            'job_description': self.job_description,
            'skills': self.skills
        }

class Posting(db.Model):
    __tablename__ = 'posting'
    posting_id = db.Column(db.Integer, primary_key=True,
        autoincrement=True)
    title = db.Column(db.String(255), nullable=False)
    posting_description = db.Column(db.Text, nullable=True)
    job_posting_url = db.Column(db.String(255), nullable=True)
    application_type = db.Column(db.String(100), nullable=True)
    skills_description = db.Column(db.Text, nullable=True)
    formatted_worktype = db.Column(db.String(100), nullable=True)
    zip_code = db.Column(db.String(10), nullable=True)
    remote_allowed = db.Column(db.Boolean, nullable=True)
    location = db.Column(db.String(255), nullable=True)
    company_id = db.Column(db.BigInteger, db.ForeignKey('company.
        company_id'), nullable=False)
    job_id = db.Column(db.BigInteger, db.ForeignKey('job.job_id'),
        nullable=False)
    states = db.relationship('PostingState', backref='posting',
        cascade="all, delete-orphan", lazy=True)
    additional_infos = db.relationship('AdditionalInfo', backref='
        posting', cascade="all, delete-orphan", lazy=True)
    def to_dict(self):
        return {
            "posting_id": self.posting_id,
            "title": self.title,
            "posting_description": self.posting_description,
            "job_posting_url": self.job_posting_url,
            "application_type": self.application_type,
            "skills_description": self.skills_description,
            "formatted_worktype": self.formatted_worktype,
            "zip_code": self.zip_code,
            "remote_allowed": self.remote_allowed,
            "location": self.location,
            "company_id": self.company_id,
            "job_id": self.job_id
```

```
        }

class PostingState(db.Model):
    __tablename__ = 'posting_state'
    posting_state_id = db.Column(db.Integer, primary_key=True)
    posting_id = db.Column(db.Integer, db.ForeignKey('posting.
        posting_id'), nullable=False)
    timezone_offset = db.Column(db.Integer, nullable=True)
    expiry = db.Column(db.DateTime, nullable=False)
    original_listed_time = db.Column(db.DateTime, nullable=False)
    listed_time = db.Column(db.DateTime, nullable=False)
    applies = db.Column(db.Integer, nullable=True)
    remaining_time = db.Column(db.Integer, nullable=True)
    apply_rate = db.Column(db.Float, nullable=True)
    views = db.Column(db.Integer, nullable=True)
    def to_dict(self):
        data = {
            'posting_state_id': self.posting_state_id,
            'posting_id': self.posting_id,
            'timezone_offset': self.timezone_offset,
            'expiry': self.expiry,
            'original_listed_time': self.original_listed_time,
            'listed_time': self.listed_time,
            'applies': self.applies,
            'remaining_time': self.remaining_time,
            'apply_rate': self.apply_rate,
            'views': self.views,
        }
        for field in self.__dict__:
            value = getattr(self, field)
            if isinstance(value, bytes):
                data[field] = base64.b64encode(value).decode('utf-8')
        return data

class AdditionalInfo(db.Model):
    __tablename__ = 'additional_info'
    additional_info_id = db.Column(db.Integer, primary_key=True)
    posting_id = db.Column(db.Integer, db.ForeignKey('posting.
        posting_id'), nullable=False)
    formatted_experience_level = db.Column(db.Text, nullable=True)
    posting_domain = db.Column(db.Text, nullable=True)
    application_url = db.Column(db.Text, nullable=True)
    close_time = db.Column(db.DateTime, nullable=True)
    def to_dict(self):
        data = {
            "additional_info_id": self.additional_info_id,
            "posting_id": self.posting_id,
            "formatted_experience_level": self.
                formatted_experience_level,
            "posting_domain": self.posting_domain,
            "application_url": self.application_url,
            "close_time": self.close_time
        }
```

```
        for field in self.__dict__:
            value = getattr(self, field)
            if isinstance(value, bytes):
                data[field] = base64.b64encode(value).decode('utf-8')
        return data
```

Listing 75: Các hàm chuyển đổi dữ liệu liên quan đến bài đăng **Posting** từ các bảng cơ sở dữ liệu thành định dạng từ điển.

5. **Hàm lấy danh sách bài đăng Posting từ cơ sở dữ liệu.**

```
@posting_bp.route('/posting', methods=['GET'])
def get_postings():
    page = request.args.get('page', 1, type=int)
    per_page = request.args.get('per_page', 10, type=int)
    try:
        pagination = Posting.query.order_by(Posting.posting_id).
            paginate(
            page=page, per_page=per_page, error_out=False
        )
        return jsonify({
            "postings": [posting.to_dict() for posting in pagination.
                items],
            "total": pagination.total,
            "pages": pagination.pages,
            "current_page": pagination.page
        })
    except Exception as e:
        print(f"Error in get_postings: {e}")
        return jsonify({"error": "An unexpected error occurred", "
            details": str(e)}), 500
```

Listing 76: Hàm lấy danh sách bài đăng từ cơ sở dữ liệu.

6. **Hàm lấy danh sách trạng thái các bài đăng Posting từ cơ sở dữ liệu**

```
@posting_bp.route('/posting_state', methods=['GET'])
def get_posting_states():
    page = request.args.get('page', 1, type=int)
    per_page = request.args.get('per_page', 10, type=int)
    try:
        pagination = PostingState.query.order_by(PostingState.
            posting_state_id).paginate(page=page, per_page=per_page,
            error_out=False)
        if not pagination.items:
            return jsonify({"message": "No posting states found"}),
                404

        states = [state.to_dict() if state else None for state in
            pagination.items]
        return jsonify({
            "states": states,
            "total": pagination.total,
```

```
            "pages": pagination.pages,
            "current_page": pagination.page
        })
    except Exception as e:
        error_details = traceback.format_exc()
        print(f"Error in get_posting_states: {error_details}")
        return jsonify({"error": "An unexpected error occurred", "
            details": error_details}), 500
```

Listing 77: Hàm lấy danh sách trạng thái bài đăng từ cơ sở dữ liệu.

7. **Hàm lấy thông tin thêm của các bài đăng Posting từ cơ sở dữ liệu**

```
@posting_bp.route('/additional_information', methods=['GET'])
def get_additional_information():
    try:
        infos = AdditionalInfo.query.all()
        if not infos:
            return jsonify({
                "current_page": 1,
                "pages": 1,
                "total": 1,
                "additional_information": [None]
            }), 200
        return jsonify({
            "current_page": 1,
            "pages": 1,
            "total": len(infos),
            "additional_information": [info.to_dict() for info in
                infos]
        }), 200
    except Exception as e:
        error_details = traceback.format_exc()
        print(f"Error in get_additional_information: {error_details}"
            )
        return jsonify({"error": "An unexpected error occurred", "
            details": error_details}), 500
```

Listing 78: Hàm lấy thông tin thêm của các bài đăng từ cơ sở dữ liệu.

8. **Hàm tạo bài đăng Posting mới cho cơ sở dữ liệu**

```
@posting_bp.route('/posting', methods=['POST'])
def create_posting():
    data = request.json
    required_fields = ['title', 'location', 'company_id', 'job_id']
    for field in required_fields:
        if field not in data:
            return jsonify({"error": f"Field '{field}' is required"})
                , 400
    data.pop('posting_id', None)
    try:
        new_posting = Posting(**data)
```

```
        db.session.add(new_posting)
        db.session.commit()
        return jsonify({"message": "Posting created successfully", "
            posting": new_posting.to_dict()}), 201
    except Exception as e:
        print(f"Error while creating posting: {str(e)}")
        return jsonify({"error": "An unexpected error occurred", "
            details": str(e)}), 500
```

Listing 79: Hàm tạo bài đăng mới cho cơ sở dữ liệu.

9. **Hàm cập nhật bài đăng Posting từ cơ sở dữ liệu**

```
@posting_bp.route('/posting/<int:id>', methods=['PUT'])
def update_posting(id):
    data = request.json
    posting = Posting.query.get_or_404(id)
    for key, value in data.items():
        setattr(posting, key, value)
    db.session.commit()
    return jsonify({"message": "Posting updated successfully", "
        posting": posting.to_dict()})
```

Listing 80: Hàm cập nhật bài đăng từ cơ sở dữ liệu.

10. **Hàm xóa bài đăng Posting từ cơ sở dữ liệu.**

```
@posting_bp.route('/posting/<int:id>', methods=['DELETE'])
def delete_posting(id):
    posting = Posting.query.get_or_404(id)
    db.session.delete(posting)
    db.session.commit()
    return jsonify({"message": "Posting deleted successfully"})
```

Listing 81: Hàm xóa bài đăng từ cơ sở dữ liệu.

7.2.3 Company

1. **Các API của Company**

```
http://127.0.0.1:5000/api/company
http://127.0.0.1:5000/api/company/<id>
http://127.0.0.1:5000/api/company/industries/<id>
http://127.0.0.1:5000/api/company/specialities<id>
http://127.0.0.1:5000/api/company/location<id>
http://127.0.0.1:5000/api/company/employee_count<id>
```

Listing 82: Thiết lập các công cụ và module

2. **Thiết lập các ROUTE chính**

```
company_routes = Blueprint('company_routes', __name__)
company_routes.route('/', methods=['GET'])(company_controller.
    get_all_companies)
company_routes.route('/<id>', methods=['GET'])(company_controller.
    get_company_by_id)
company_routes.route('/industries/<id>', methods=['GET'])(
    company_controller.get_industries_by_company_id)
company_routes.route('/speciality/<id>', methods=['GET'])(
    company_controller.get_speciality_by_id)
company_routes.route('/location/<id>', methods=['GET'])(
    company_controller.get_location_by_id)
company_routes.route('/employee_count/<id>', methods=['GET'])(
    company_controller.get_employee_count_by_id)
@company_routes.route('/', methods=['POST'], endpoint='create_company
    ')
@company_middleware.validate_company_data
def create_company_route():
    return company_controller.create_company()
@company_routes.route('/<id>', methods=['PUT'], endpoint='
    update_company')
@company_middleware.validate_update_company_data
def update_company_route(id):
    return company_controller.update_company(id)
company_routes.route('/<id>', methods=['DELETE'])(company_controller.
    delete_company)
```

Listing 83: Thiết lập các ROUTE chính.

3. **Thiết lập xử lý dữ liệu trước khi chạy controller-PUT**

```
def validate_update_company_data(func):
    def wrapper(*args, **kwargs):
        data = request.json
        errors = []
        company_name = data.get("company_name")
        if company_name:
            company_name = re.sub(r"[^a-zA-Z0-9 &\-|]", "",
                company_name)
            company_name = re.sub(r"\s+", " ", company_name).strip()
            data["company_name"] = company_name
        description = data.get("description")
        if description:
            description = re.sub(r"[\n\r\t]", " ", description)
            description = re.sub(r"\s+", " ", description).strip()
            data["description"] = description
        url = data.get("company_url")
        if url:
            url_pattern = re.compile(
                r"^(https?://)?([\da-z.-]+)\.([a-z.]{2,6})([/?].*)?$"
                    , re.IGNORECASE
            )
            if not url_pattern.match(url):
                errors.append("Invalid URL format.")
```

```
        state = data.get("state")
        if state:
            state = state.strip().upper()
            if len(state) != 2 and not re.match(r"[A-Za-z]+", state):
                errors.append("State contains invalid characters or
                    format.")
        country = data.get("country")
        if country:
            errors.append("Country is required and must be in ISO 2-
                letter format.")
        city = data.get("city")
        if city:
            city = re.sub(r"[^A-Za-z\s,/-]", "", city.strip())
            if not city:
                errors.append("City contains invalid characters or is
                     empty.")
        zip_code = data.get("zip_code")
        if zip_code:
            if not re.match(r"^[A-Za-z0-9\s-]+$", zip_code.strip()):
                errors.append("Zip code is invalid.")
        number = data.get("number")
        if number:
            if not re.match(r"^\d+(-\d+)*$", number.strip()):
                errors.append("Number (street number) is invalid.")
        street = data.get("street")
        if street:
            if not re.match(r"[A-Za-z0-9\s,:\-+]+", street.strip()):
                errors.append("Street name contains invalid
                    characters.")
        industry_name = data.get("industry_name")
        if industry_name:
            if not re.match(r"^[A-Za-z0-9\s,:\-]+$", industry_name.
                strip()):
                errors.append("Industry name contains invalid
                    characters.")
        speciality = data.get("speciality")
        if speciality:
            speciality = re.sub(r"[^a-zA-Z0-9,;/&\s]", "", speciality
                )
            speciality = re.sub(r"\s+", " ", speciality).strip()
            data["speciality"] = speciality
        follower_count = data.get("follower_count")
        employee_count = data.get("employee_count")
        if (follower_count or employee_count) and not (follower_count
             and employee_count):
            errors.append("both must exist.")
        if errors:
            return jsonify({"success": False, "errors": errors}), 400

        return func(*args, **kwargs)
    return wrapper
```

Listing 84: Thiết lập xử lí dữ liệu trước khi chạy controller-PUT.

4. **Các hàm lấy danh sách dữ liệu toàn bộ công ty và theo company_id**

```
def get_all_companies():
    try:
        with engine.connect() as connection:
            query = text('SELECT TOP 100 * FROM Company')
            result = connection.execute(query)
            companies = [dict(row._mapping) for row in result]

        return jsonify(
                {
                "success": True,
                "data": companies
                }
            )
    except Exception as e:
        return jsonify({"success": False, "message": str(e)})

def get_company_by_id(id):
    try:
        with engine.connect() as connection:
            query = text(f'SELECT * FROM Company WHERE Company_id = {
                id}')
            result = connection.execute(query)
            company = [dict(row._mapping) for row in result]

        return jsonify(
                {
                "success": True,
                "data": company
                }
            )
    except Exception as e:
        return jsonify({"success": False, "message": str(e)})

def get_company_by_id(id):
    try:
        with engine.connect() as connection:
            query = text(f'SELECT * FROM Company WHERE Company_id = {
                id}')
            result = connection.execute(query)
            company = [dict(row._mapping) for row in result]

        return jsonify(
                {
                "success": True,
                "data": company
                }
            )
    except Exception as e:
        return jsonify({"success": False, "message": str(e)})
```

Listing 85: Hàm lấy danh sách toàn bộ công ty và theo company_id.

5. **Hàm lấy danh sách industries, specialities, location và employee_count theo company_id**

```
def get_industries_by_company_id(id):
    try:
        with engine.connect() as connection:
            query = text(f"""
                SELECT ci.Company_id, ci.Industry_id, i.Industry_name
                FROM Company_Has_Industry ci
                JOIN Industry_Has_Industry_Name i ON ci.Industry_id =
                    i.Industry_id
                WHERE ci.Company_id = {id}
            """)
            result = connection.execute(query)
            industries = [dict(row._mapping) for row in result]
        return jsonify(
                {
                "success": True,
                "data": industries
                }
            )
    except Exception as e:
        return jsonify({"success": False, "message": str(e)})

def get_speciality_by_id(id):
    try:
        with engine.connect() as connection:
            query = text(f'''
                SELECT chs.company_id, chs.speciality_id, cs.
                    speciality
                FROM Company_Has_Speciality AS chs
                JOIN Company_Speciality AS cs
                ON chs.speciality_id = cs.speciality_id
                WHERE chs.company_id = {id}
            ''')
            result = connection.execute(query)
            speciality = [dict(row._mapping) for row in result]
        return jsonify(
                {
                "success": True,
                "data": speciality
                }
            )
    except Exception as e:
        return jsonify({"success": False, "message": str(e)})

def get_location_by_id(id):
    try:
        with engine.connect() as connection:
            query = text(f'SELECT * FROM Company_Location WHERE
                Company_id = {id}')
            result = connection.execute(query)
            location = [dict(row._mapping) for row in result]
        return jsonify(
```

```
                {
                "success": True,
                "data": location
                }
            )
    except Exception as e:
        return jsonify({"success": False, "message": str(e)})

def get_employee_count_by_id(id):
    try:
        with engine.connect() as connection:
            query = text(f'SELECT * FROM Employee_Count WHERE
                Company_id = {id}')
            result = connection.execute(query)
            employee_count = [dict(row._mapping) for row in result]
        return jsonify(
                {
                "success": True,
                "data": employee_count
                }
            )
    except Exception as e:
        return jsonify({"success": False, "message": str(e)})
```

Listing 86: Hàm lấy danh sách industries, specialities, location và employee_count theo company_id.

6. **Hàm tạo mới công ty Company từ cơ sở dữ liệu**

```
def create_company():
    try:
        data = request.get_json()
        company_id = data['company_id']
        company_size = data['company_size']
        description = data['description']
        company_name = data['company_name']
        company_url = data['company_url']
        with engine.connect() as connection:
            query = text(f'''
                INSERT INTO Company (Company_id, Company_size,
                    description, Company_name, Company_url)
                VALUES ({company_id}, {company_size}, '{description
                    }', '{company_name}', '{company_url}')
                COMMIT;
            ''')
            connection.execute(query)
            if 'country' in data:
                country = data['country']
                state = data.get('state', None)
                city = data.get('city', None)
                zip_code = data.get('zip_code', None)
                number = data.get('number', None)
                street = data.get('street', None)
                location_query = text(f'''
```

```
                    INSERT INTO company_location (company_id, country
                        , state, city, zip_code, number, street)
                    VALUES ({company_id}, '{country}', {f"'{state}'"
                        if state else "0"}, {f"'{city}'" if city else
                         "0"}, {f"'{zip_code}'" if zip_code else "
                        NULL"}, {f"'{number}'" if number else "NULL
                        "}, {f"'{street}'" if street else "NULL"})
                    COMMIT;
                ''')
                connection.execute(location_query)
            if 'industry_id' in data:
                industry_query = text(f'''
                    INSERT INTO Company_Has_Industry (company_id,
                        industry_id)
                    VALUES ({company_id}, {data['industry_id']})
                    COMMIT;
                ''')
                connection.execute(industry_query)
            elif 'industry_name' in data:
                get_industry_query = text(f'''
                    SELECT industry_id FROM
                        Industry_Has_Industry_Name WHERE
                        industry_name = '{data['industry_name']}'
                ''')
                result = connection.execute(get_industry_query).
                    fetchone()
                if result:
                    industry_id = result[0]
                else:
                    insert_industry_query = text(f'''
                        INSERT INTO Industry_Has_Industry_Name (
                            industry_name)
                        VALUES ('{data['industry_name']}')
                        COMMIT;
                    ''')
                    connection.execute(insert_industry_query)
                    industry_id = connection.execute(
                        get_industry_query).fetchone()[0]
                industry_insert_query = text(f'''
                    INSERT INTO Company_Has_Industry (company_id,
                        industry_id)
                    VALUES ({company_id}, {industry_id})
                    COMMIT;
                ''')
                connection.execute(industry_insert_query)
            if 'speciality_id' in data:
                speciality_query = text(f'''
                    INSERT INTO Company_Has_Speciality (company_id,
                        speciality_id)
                    VALUES ({company_id}, {data['speciality_id']})
                    COMMIT;
                ''')
                connection.execute(speciality_query)
```

```
            elif 'speciality' in data:
                get_speciality_query = text(f'''
                    SELECT speciality_id FROM Company_Speciality
                        WHERE speciality = '{data['speciality']}'
                ''')
                result = connection.execute(get_speciality_query).
                    fetchone()
                if result:
                    speciality_id = result[0]
                else:
                    insert_speciality_query = text(f'''
                        INSERT INTO Company_Speciality (speciality)
                        VALUES ('{data['speciality']}')
                        COMMIT;
                    ''')
                    connection.execute(insert_speciality_query)

                    speciality_id = connection.execute(
                        get_speciality_query).fetchone()[0]
                speciality_insert_query = text(f'''
                    INSERT INTO Company_Has_Speciality (company_id,
                        speciality_id)
                    VALUES ({company_id}, {speciality_id})
                    COMMIT;
                ''')
                connection.execute(speciality_insert_query)
            if 'employee_count' in data:
                employee_count_query = text(f'''
                    INSERT INTO Employee_Count (Company_id,
                        Employee_count, Follower_count, Time_recorded
                        )
                    VALUES ({company_id}, {data['employee_count']}, {
                        data['follower_count']}, {current_timestamp})
                    COMMIT;
                ''')
                connection.execute(employee_count_query)
        return jsonify({
            "success": True,
            "message": "Company created successfully"
        })
    except Exception as e:
        return jsonify({"success": False, "message": str(e)})
```

Listing 87: Hàm tạo công ty **Company** mới từ cơ sở dữ liệu.

7. **Hàm cập nhật công ty Company từ cơ sở dữ liệu**

```
def update_company(id):
    try:
        data = request.get_json()
        company_size = data.get('company_size', None)
        description = data.get('description', None)
        company_name = data.get('company_name', None)
        company_url = data.get('company_url', None)
```

```
        set_clause = []
        if company_size is not None:
            set_clause.append(f"Company_size = {company_size}")
        if description is not None:
            set_clause.append(f"description = '{description}'")
        if company_name is not None:
            set_clause.append(f"Company_name = '{company_name}'")
        if company_url is not None:
            set_clause.append(f"Company_url = '{company_url}'")
        if set_clause:
            update_query = text(f'''
                UPDATE Company
                SET {', '.join(set_clause)}
                WHERE Company_id = {id}
                COMMIT;
            ''')
            with engine.connect() as connection:
                connection.execute(update_query)
        if any(key in data for key in ['country', 'state', 'city', '
            zip_code', 'number', 'street']):
            fields_to_update = []
            if 'country' in data:
                fields_to_update.append(f"country = '{data['country
                    ']}'")
            if 'state' in data:
                fields_to_update.append(f"state = '{data['state']}'"
                    if data['state'] else "state = 0")
            if 'city' in data:
                fields_to_update.append(f"city = '{data['city']}'" if
                     data['city'] else "city = 0")
            if 'zip_code' in data:
                fields_to_update.append(f"zip_code = '{data['zip_code
                    ']}'" if data['zip_code'] else "zip_code = NULL")
            if 'number' in data:
                fields_to_update.append(f"number = '{data['number']}'
                    " if data['number'] else "number = NULL")
            if 'street' in data:
                fields_to_update.append(f"street = '{data['street']}'
                    " if data['street'] else "street = NULL")

            if fields_to_update:
                update_query =text(f'''
                    UPDATE company_location
                    SET {', '.join(fields_to_update)}
                    WHERE company_id = {id};
                    COMMIT;
                ''')
                with engine.connect() as connection:
                    connection.execute(update_query)
        if 'industry_id' in data or 'industry_new_id' in data or '
            industry_new_name' in data:
            with engine.connect() as connection:
                if 'industry_new_id' in data:
```

```
                    industry_new_id = data['industry_new_id']
                    update_query = text(f'''
                        UPDATE Company_Has_Industry
                        SET industry_id = {industry_new_id}
                        WHERE company_id = {id} AND industry_id = {
                            data.get('industry_id', 0)};
                        COMMIT;
                    ''')
                    connection.execute(update_query)
                elif 'industry_new_name' in data:
                    industry_new_name = data['industry_new_name']
                    select_query = text(f'''
                        SELECT industry_id FROM
                            Industry_Has_Industry_Name
                        WHERE industry_name = '{industry_new_name}';
                    ''')
                    result = connection.execute(select_query).
                        fetchone()
                    if result:
                        industry_new_id = result[0]
                        update_query = text(f'''
                            UPDATE Company_Has_Industry
                            SET industry_id = {industry_new_id}
                            WHERE company_id = {id} AND industry_id =
                                {data.get('industry_id', 0)};
                            COMMIT;
                        ''')
                        connection.execute(update_query)
    if 'speciality_id' in data or 'speciality_new_id' in data or
        'speciality_new_name' in data:
        with engine.connect() as connection:
            if 'speciality_new_id' in data:
                speciality_new_id = data['speciality_new_id']
                update_query = text(f'''
                    UPDATE Company_Has_Speciality
                    SET speciality_id = {speciality_new_id}
                    WHERE company_id = {id} AND speciality_id = {
                        data.get('speciality_id', 0)};
                    COMMIT;
                ''')
                connection.execute(update_query)
            elif 'speciality_new_name' in data:
                speciality_new_name = data['speciality_new_name']
                select_query = text(f'''
                    SELECT speciality_id FROM Company_Speciality
                    WHERE speciality = '{speciality_new_name}';
                ''')
                result = connection.execute(select_query).
                    fetchone()
                if result:
                    speciality_new_id = result[0]
                    update_query = text(f'''
                        UPDATE Company_Has_Speciality
```

```
                            SET speciality_id = {speciality_new_id}
                            WHERE company_id = {id} AND speciality_id
                                 = {data.get('speciality_id', 0)};
                            COMMIT;
                        ''')
                        connection.execute(update_query)
        if 'time_recorded' in data:
            with engine.connect() as connection:
                update_query = text(f'''
                    UPDATE Employee_Count
                    SET Employee_count = {data['employee_count']},
                        Follower_count = {data['follower_count']},
                        time_recorded = {current_timestamp}
                    WHERE Company_id = {id} AND Time_recorded = {data
                        ['time_recorded']}
                    COMMIT;
                ''')
                connection.execute(update_query)
        return jsonify({
            "success": True,
            "message": "Company updated successfully"
        })
    except Exception as e:
        return jsonify({"success": False, "message": str(e)})
```

Listing 88: Hàm cập nhật công ty **Company** từ cơ sở dữ liệu.

8. **Hàm xóa công ty Company từ cơ sở dữ liệu**

```
def delete_company(id):
    try:
        with engine.connect() as connection:
            delete_industry_query = text(f'''
                DELETE FROM Company_Has_Industry WHERE company_id = {
                    id}
                COMMIT;
            ''')
            connection.execute(delete_industry_query)
            delete_location_query = text(f'''
                DELETE FROM company_location WHERE company_id = {id}
                COMMIT;
            ''')
            connection.execute(delete_location_query, {'company_id':
                id})
            delete_speciality_query = text(f'''
                DELETE FROM Company_Has_Speciality WHERE company_id =
                     {id}
                COMMIT;
            ''')
            connection.execute(delete_speciality_query)
            delete_employee_count_query = text(f'''
                DELETE FROM Employee_Count WHERE company_id = {id}
                COMMIT;
            ''')
```

```
            connection.execute(delete_employee_count_query)
            delete_company_query = text(f'''
                DELETE FROM Company WHERE company_id = {id}
                COMMIT;
            ''')
            connection.execute(delete_company_query)
        return jsonify({
            "success": True,
            "message": "Company and related data deleted successfully
                "
        })
    except Exception as e:
        return jsonify({"success": False, "message": str(e)})
```

Listing 89: Hàm xóa công ty **Company** từ cơ sở dữ liệu.

7.2.4 Dashboard

1. **Các API của Dashboard**

```
http://127.0.0.1:5000/worktype_count
http://127.0.0.1:5000/api/v1/salary_distribution_by_location?location
    =${location}
http://127.0.0.1:5000/posting_title
http://127.0.0.1:5000/api/v1/job_title_frequency_by_location?location
    =${location}
http://127.0.0.1:5000/api/company/chart_company_postings/<id>
```

Listing 90: Thiết lập các công cụ và module.

2. **Thiết lập các công cụ và module**

```
from flask import Blueprint, jsonify, request
from app.models import db, Posting, Salary, Salary_Type
from sqlalchemy import func
```

Listing 91: Thiết lập các công cụ và module.

3. **Tạo Blueprint tên Posting để tổ chức các route.**

```
dashboard_bp = Blueprint('dashboard', __name__)
```

Listing 92: Tạo Blueprint tên Posting để tổ chức các route trong Flask.

4. **Hàm thống kê tỉ lệ worktype**

```
@GN_bp.route('/worktype_count', methods=['GET'])
def worktype_count():
    try:
        with engine.connect() as connection:
            result = connection.execute(text("SELECT
                formatted_worktype, count(*) as count FROM posting
                GROUP BY formatted_worktype"))
```

```
            rows = [dict(row._mapping) for row in result]
            return jsonify(
                {
                    "success": True,
                    "data": rows
                }
            )
    except Exception as e:
        return jsonify(
            {
                "success": False,
                "message": str(e)
            }
        )
```

Listing 93: Hàm thống kê tỉ lệ worktype

5. **Hàm lấy ra top 15 từ khóa xuất hiện nhiều nhất trong posting title**

```
@GN_bp.route('/posting_title', methods=['GET'])
def posting_title():
    try:
        with engine.connect() as connection:
            result = connection.execute(text("SELECT TOP 3000 title
                FROM posting"))
            rows = [dict(row._mapping) for row in result]
            titles = [job["title"].replace("\r\n", "").strip() for
                job in rows]
            stopwords = {"or", "and", "else", "the", "in", "on", "at"
                , "for", "to", "a", "an", "of", "with"}
            words = []
            for title in titles:
                for word in re.findall(r'\b\w+\b', title.lower()):
                    if word not in stopwords:
                        words.append(word)
            word_counts = Counter(words)
            top_keywords = word_counts.most_common(15)
            return jsonify(
                {
                    "success": True,
                    "data": top_keywords
                }
            )
    except Exception as e:
        return jsonify(
            {
                "success": False,
                "message": str(e)
            }
        )
```

Listing 94: Hàm Hàm lấy ra top 15 từ khóa xuất hiện nhiều nhất trong posting title

6. **Hàm lấy tần suất xuất hiện của các chức danh công việc theo vị trí, trả về 20 kết quả phổ biến nhất từ cơ sở dữ liệu**

```
@dashboard_bp.route('/job_title_frequency_by_location', methods=['GET
    '])
def job_title_frequency_by_location():
    location = request.args.get('location', '').strip()
    if not location:
        return jsonify({"error": "Location parameter is required"}),
            400
    query = db.session.query(
        Posting.title,
        Posting.location,
        func.count().label('frequency')
    ).filter(Posting.location.ilike(f"%{location}%"))
    query = query.group_by(Posting.title, Posting.location) \
                 .order_by(func.count().desc()) \
                 .limit(20) \
                 .all()
    result = [{
        "title": row.title,
        "location": row.location,
        "frequency": row.frequency
    } for row in query]
    if not result:
        return jsonify({"message": "No job postings found for the
            given location"}), 404
    return jsonify(result)
```

Listing 95: Hàm lấy tần suất xuất hiện của các chức danh công việc theo vị trí, trả về 20 kết quả phổ biến nhất từ cơ sở dữ liệu.

7. **Hàm phân bố lương trung bình theo vị trí và chu kỳ trả lương từ cơ sở dữ liệu**

```
from sqlalchemy import case, func
@dashboard_bp.route('/salary_distribution_by_location', methods=['GET
    '])
def salary_distribution_by_location():
    location = request.args.get('location', '').strip()
    if not location:
        print("Location is missing!")
        return jsonify({"error": "Location parameter is required"}),
            400

    print(f"Location received: {location}")
    try:
        query = db.session.query(
            Posting.location,
            Salary.pay_period,
            func.avg(
                case(
                    (Salary_Type.salary_type == 'min', Salary_Type.
                        value),
                    else_=None
```

```
                )
            ).label('avg_min_salary'),
            func.avg(
                case(
                    (Salary_Type.salary_type == 'med', Salary_Type.
                        value),
                    else_=None
                )
            ).label('avg_med_salary'),
            func.avg(
                case(
                    (Salary_Type.salary_type == 'max', Salary_Type.
                        value),
                    else_=None
                )
            ).label('avg_max_salary')
        ).join(Salary_Type, Salary.salary_id == Salary_Type.salary_id
            ) \
         .filter(Posting.location.ilike(f"%{location}%")) \
         .group_by(Posting.location, Salary.pay_period) \
         .order_by(Posting.location) \
         .all()
        print("Query executed successfully.")
        result = [{
            "location": row.location,
            "pay_period": row.pay_period,
            "avg_min_salary": row.avg_min_salary,
            "avg_med_salary": row.avg_med_salary,
            "avg_max_salary": row.avg_max_salary
        } for row in query]
        if not result:
            print("No salary data found for the given location.")
            return jsonify({"message": "No salary data found for the
                given location"}), 404
        return jsonify(result)
    except Exception as e:
        print(f"Error occurred: {str(e)}")
        return jsonify({"error": f"An internal server error occurred:
            {str(e)}"}), 500
```

Listing 96: Hàm phân bố lương trung bình (bao gồm mức lương tối thiểu, trung bình, tối đa) theo vị trí và chu kỳ trả lương từ cơ sở dữ liệu.

8. **Hàm lượng bài đăng của công ty trong tháng 4/2024**

```
company_routes.route('/chart_company_postings/<id>', methods=['GET'])
    (company_controller.get_chart_company_postings)
def get_chart_company_postings(id):
    try:
        with engine.connect() as connection:
            query = text(f"""
                SELECT
                    ps.original_listed_time,
                    COUNT(*) AS post_count
```

```
                FROM Posting_State ps
                JOIN Posting p ON ps.posting_state_id = p.posting_id
                WHERE p.company_id = {id}
                GROUP BY ps.original_listed_time
                ORDER BY ps.original_listed_time
            """)
            result = connection.execute(query)
            chart_data = [dict(row._mapping) for row in result]
        return jsonify(
                {
                "success": True,
                "data": chart_data
                }
            )
    except Exception as e:
        return jsonify({"success": False, "message": str(e)})
```

Listing 97: **Route** và **Controller** lượng bài đăng của công ty trong tháng 4/2024.

8 Kết luận

Trong đồ án này, chúng em đã triển khai thành công một hệ thống quản lý nội dung (Content Management System) tích hợp xử lý dữ liệu lớn từ nền tảng Kaggle. Thông qua việc thiết kế lại mô hình ERD, tối ưu cơ sở dữ liệu với các trigger và index, chúng em đảm bảo hiệu năng xử lý và truy xuất dữ liệu hiệu quả trên SQL Server. Ngoài ra, việc tích hợp Flask và ReactJS đã tạo nên một hệ thống web hiện đại, trực quan, hỗ trợ quản lý và báo cáo dữ liệu với các biểu đồ chi tiết. Đồ án không chỉ đáp ứng yêu cầu kỹ thuật mà còn mở ra hướng ứng dụng thực tế trong các hệ thống quản lý nội dung quy mô lớn. Chúng em xin chân thành cảm ơn giảng viên hướng dẫn vì những định hướng và hỗ trợ quý báu trong suốt quá trình thực hiện đồ án. Những ý kiến đóng góp và sự hỗ trợ tận tình của giảng viên đã giúp chúng em hoàn thiện dự án một cách tốt nhất.