SAN JOSE STATE UNIVERSITY

Database Systems for Analytics (DATA 225)

Instructor – Simon Shim

Report on

LinkedIn Job Posting Analysis

Group 8

Submitted By

Aiswarya Raghavadesikan

Neha Thakur

Saumya Varshney

Shikha Singh

Venkata Sai Sreelekha Gollu

TABLE OF CONTENT

I. Introduction

The project focuses on analyzing LinkedIn job postings, offering valuable insights into the ever-changing job market. LinkedIn is an employment focused social media platform which caters to the requirements of both employers and employees. With the dynamic job market, LinkedIn is offering insights into the skills necessary to excel in the contemporary professional world. With over 800 million users worldwide, LinkedIn provides a vast source of data on job opportunities. Each individual posting contains several valuable attributes, including the title, job description, salary, location, application URL, and work-types (remote, contract, etc.), in addition to separate files containing the benefits, skills, and industries associated with each posting.

This project aims to extract meaningful information, including job trends by industry, location, skills in demand, and changes over time. LinkedIn Database offers the distribution of job postings across various companies and industries. Skill distribution and emerging job roles could provide great insights into the changing and evolving needs of companies. This database helps to tackle real-world employment challenges, ultimately enhancing our understanding of today's job market and assisting professionals in making informed business decisions.

II. Problem Statement

The analysis and management of the LinkedIn job postings dataset pose a significant challenge due to the vast and dynamic nature of the data. This dataset encompasses a diverse range of job categories, industries, and geographic locations, making it challenging to extract meaningful insights and trends. Key issues include the identification of emerging job roles, the assessment of skill sets in demand, the determination of competitive salary ranges, and the discernment of dynamic market trends. Consequently, addressing these challenges is critical to enabling informed decision-making in the job market and enhancing the effectiveness of job search and talent acquisition processes.

III. Solution Requirement

In order to efficiently understand the dataset, the first step is to clean the given dataset to more meaningful data.

Elucidating a given job posting will immensely help a job seeker to further understand the company and the current job market trends. Our data insights provides a job seeker with quick and valuable information regarding an active job posting, its associated salary range, work mode and work type. Analyzing the filtered data will help a job seeker to make an efficient decision based on the dynamic job market trends.

Job companies and recruiters can also simultaneously benefit from this which includes viewing the number of applicants for a given job posting, identifying the potential candidates, closing out the job posting promptly.

Functional analysis helps in processing the given dataset by fetching the appropriate and relevant information from the table and delivering profound insights to the job poster and a job seeker. However, the recruiter's contact information, current employers of a company for internal referrals, job seeker information are some of the aspects which are out of the scope of this project.

IV. Limitations

Incomplete Data: This LinkedIn job postings dataset may not provide a comprehensive overview of the job market due to unlisted or private job listings. The dataset may lack diversity in terms of industries, regions, and job types.

Limited Historical Data Access to historical job postings data on LinkedIn can be restricted, making it challenging to conduct long-term studies or track trends over time. Job postings can change rapidly, affecting the accuracy of historical analysis.

Privacy Concerns: Handling LinkedIn data raises privacy concerns. Respecting the privacy of individuals and organizations within the dataset and adhering to LinkedIn's terms of service is crucial to maintain ethical data analysis practices.

Frequency of Data Updates: Frequent updates and deletions of data on LinkedIn necessitate investigation into how often the data can be updated and retrieved. Regular updates are crucial for maintaining the relevance and accuracy of the analysis.

In some cases, company details are not present completely corresponding to job postings. This might lead to inaccurate analysis for companies.

This dataset lacks information about jobseekers. Due to which it is difficult to draw insights for aspiring candidates.

V. Conceptual Database Design

To implement LinkedIn job posting analysis, Here are some of the key entities mentioned which will be helpful in drawing meaning full insights.

- Job Postings
- Job industries
- Job skills
- Benefits
- Companies
- Company Specialties
- Company Industries
- Employee counts

- **Job Postings:** It is a central repository for job listings and is commonly used in job search platforms. The attributes used in this table are job_id , company_id , title, description, skills_desc, work_type, location, currency, remote_allowed, sponsored, max_salary, med_salary, min_salary, pay_period, compensation_type, formatted_work_type, formatted_experience_level, applies, views, original_listed_time, listed_time, expiry, closed_time, posting_domain, job_posting_url, application_url , application_type

  PRIMARY KEY: job_id FOREIGN KEY: company_id
- **Job industries:** The job_industries table is used to associate job postings with specific industries. It serves as a bridge between job postings and industries, allowing

users to categorize job listings based on the industry they belong to. The attributes used in this table are, job_id, industry_id We are creating composite primary key with (job_id,industry_id) Foreign key: job_id

- **Job skills:** This table contains information about the skills required for various job postings. The attributes in this table are, job_id, skill_abr We are creating composite primary key with (job_id, skill_abr) Foreign key: job_id
- **Benefits:** The benefits table stores data related to benefits offered by companies in job postings. It includes information about the types of benefits (e.g., 401K, Medical Insurance) and whether these benefits are explicitly tagged or inferred from the job posting text. The attributes in this table are, job_id, type, inferred We are creating composite primary key with (job_id, type) Foreign key: job_id
- **Companies:**The companies table contains data about various companies and serves as a reference for job postings. It includes details about the company's name, description, size, location, and more. This table is typically linked to the job_postings table to associate job postings with specific companies. The attributes in this table are, company_id, name, description, company_size, address, city, state, country, zip_code, url. Primary key: company_id
- **Employee counts:** This table tracks the number of employees at each company. It often includes information about the company's follower count on a platform. Employee counts are crucial for understanding the size and workforce of different companies. The attributes in this table are, company_id, time_recorded, employee_count, follower_count.
  We are creating composite primary key with company_id, time_recorded, employee_count, follower_count Foreign key: company_id
- **Company specialities:** This table associates companies with their specializations or areas of expertise. The attributes in this table are, company_id, speciality. We are creating composite primary key with company_id, speciality Foreign key: company_id
- **Company_industries:** The company_industries table links companies with the industries they are associated with. The attributes in this table are company_id, industry Primary key: company_id Foreign key: company_id

A. *Data Cleaning*

As part of the data cleaning process, we inspected each table for duplicate rows and identified columns which needed to be imputed.

*1) **Duplicates**:* We cannot accurately identify primary keys in a table if there are duplicate values present.

- **job_skills** - There were no duplicate rows.
- **job_industries** - There were no duplicate rows.
- **benefits** - There were no duplicate rows.

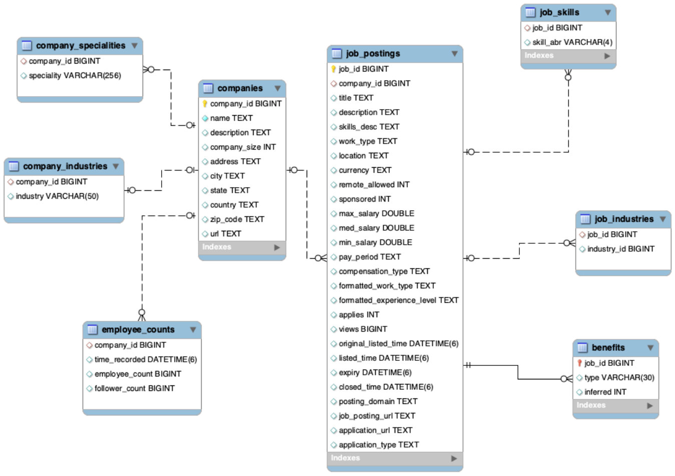


Fig. 1. ER Diagram for Raw Data

- **job_postings** - There were no duplicate rows.
- **companies** - There were no duplicate rows.
- **employee_counts** - There were 3356 duplicate rows.
- **company_industries** - There were 9877 duplicate rows.
- **company_specialities** - There were 85844 duplicate rows.

*2) **Imputation**:* Imputation for null values: Imputation is the process of replacing missing values with substituted data. It is done as a pre-processing step.If the data is numerical then we can use mean and median or mode values. Here in the data we mostly used Median for numerical values as it is robust to outliers. If the data is categorical, we can use mode which is a frequently occurring value. Here imputation is implemented on Job_postings and companies tables in order to remove null values according to the required conditions.

- **job_skills** - There was no imputation done on this table.
- **job_industries** - There was no imputation done on this table.
- **benefits** - There was no imputation done on this table.
- **job_postings** - We imputed currency, pay_period column with the mode value and applies, views, max_salary, med_salary, min_salary with median value.
- **companies** - We imputed the company_size with mode of this column.
- **employee_counts** - There was no imputation done on this table.
- **company_industries** - There was no imputation done on this table.
- **company_specialities** - There was no imputation done on this table.

B. *Enforcing constraints*

As you can see in Fig. 1 before we clean the data, the ER diagram for this raw dataset has many strong entities and none of the relationships are deterministic. For example, lets consider companies and company_specialities. Without data cleaning company_specialities table has company_ids that do not exist in the companies table. Also, not all company_ids from the companies table were present in the company_specialities table. Upon removing duplicate rows and company_ids that do not exist from company_specialities we make company_specialities a weak entity of companies. We followed the same process for all other tables and ended up with only two strong entities companies and job_postings making all other entities as weak entities of either companies or job_postings.

All the rows mentioned below were violating foreign key constraints from the parent table. If these rows are not removed, the data will be inconsistent in the these tables, as the foreign key will be pointing to IDs that are not present in the parent table.

- **job_skills** - There were 932 rows with job_id that were not present in job_postings.
- **job_industries** - There are 1040 rows with job_id that were not present in job_postings.
- **benefits** - There are no rows where the job_id is not present in job_postings.
- **job_postings** - There were 50 company_ids in job_postings that didn't exist in companies table. Since a job_posting can exist without a company, we replaced these 50 company_ids with NULL.
- **companies** - There are no foreign keys in companies.
- employee_counts - There were 50 rows with company_id that were not present in companies.
- **company_industries** - There were 50 rows with company_id that were not present in companies.
- **company_specialities** - There were 384 rows with company_id that were not present in companies.

By enforcing the above constraints, we can also identify the weak and strong entities in the database now -

- Strong Entities - job_postings, companies
- Weak Entities of job_postings - benefits, job_industries, job_skills
- Weak Entities of companies - company_industries, company_specialities, employee_counts

C. *Normalization*

Database normalization is done to reduce data redundancy and remove data inconsistency in the database. After removing all the rows which were referring to non-existent job_ids and company_ids and removing duplicate rows in our database all of our relations are in 3 NF (i.e., no multi-valued attributes, no partial dependencies, no transitive dependencies) except for company_specialities. The speciality attribute in company_specialities was multi-valued. This violates the 1 NF constraint. So, we split all the rows with multi-valued attributes in this relation in to separate rows with company_id and speciality forming a composite primary key. Now this company_specialities relation is in 3 NF.

The relations in this database are as follows:

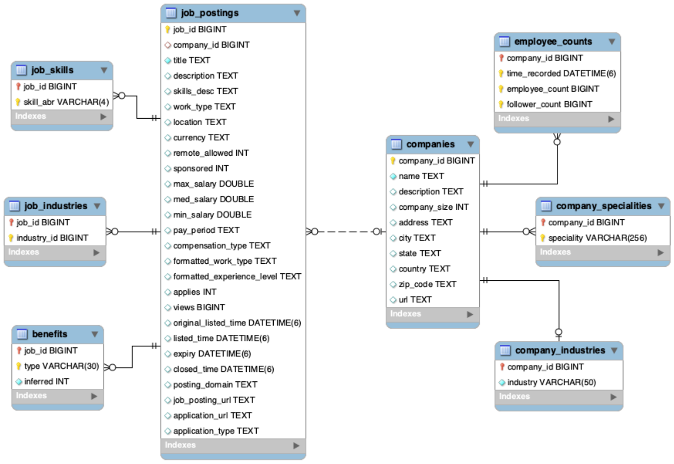


Fig. 2. ER Diagram for Clean Data

- **companies**(company_id, name, description, company_size, address, city, state, country, zip_code, url)
- **company_industries**(company_id, industry)
- **company_specialities**(company_id, **speciality**)
- **employee_counts**(company_id, time_recorded, employee_count, follower_count)
- **job_postings**(job_id, company_id, title, description, skills_desc, work_type, location, currency, remote_allowed, sponsored, max_salary, med_salary, min_salary, pay_period, compensation_type, formatted_work_type, formatted_experience_level, applies, views, original_listed_time, listed_time, expiry, closed_time, posting_domain, job_posting_url, application_url, application_type)
- **job_skills**(job_id, skill_abr)
- **job_industries**(job_id, industry_id)
- **benefits**(job_id, type, inferred)

VI. Functional Analysis and Components

Linkedin Job posting analysis system can be used by a wide range of users, including job seekers, data analysts, researchers, and HR representatives. Its various functionalities can be accessed by different individuals. Following are details explaining how it works for different people:

- Job seekers can refine their job search by considering factors such as the compensation structure (Hourly, Monthly, or Yearly) and the salary range, enabling them to make well-informed decisions when applying for positions. In addition, job seekers are empowered with a versatile array of filters, allowing them to tailor their search based on work arrangements (Full-Time, Part-Time, Internship, or Contract) and the spectrum of benefits offered by the organizations.
- Detailed insights into the companies, including their core competencies and industry involvement, provide job seekers with a comprehensive understanding of potential employers.

- Human Resources (HR) representatives can conduct a comprehensive analysis of analogous job listings offered by various organizations. This analysis encompasses an evaluation of the requisite skill sets and qualifications necessary for the creation of new job postings.
- Data analyst/Researcher can analyze emerging job roles and the requisite skill sets that are in high demand. Furthermore, they undertake the evaluation of prevalent salary ranges offered by various companies in the job market.

VII. SQL CODE SNIPPETS/QUERIES

A. *Queries*

1) Job Categorization based on industries, companies and job functions.

```
SELECT JI.industry_id, C.company_id, JP.job_id, C.name AS ”company name”,
CI.industry, JS.skill_abr,
JP.title AS job_designation
FROM job_postings AS JP
INNER JOIN job_industries AS JI ON JP.job_id = JI.job_id
INNER JOIN job_skills AS JS ON JP.job_id = JS.job_id
INNER JOIN companies AS C ON C.company_id=JP.company_id
INNER JOIN company_industries AS CI ON JP.company_id=CI.company_id
ORDER BY JI.industry_id,C.company_id, JP.job_id;
```

2) Geographical distribution of Benefits and job roles provided by companies.

```
SELECT C.company_id, JP.job_id, C.country, C.name AS ’company name’,
C.city, C.country, C.state, JP.title,
JP.formatted_experience_level, JP.work_type, B.type AS benefit_type FROM
companies C
INNER JOIN job_postings AS JP ON C.company_id = JP.company_id
INNER JOIN benefits AS B ON JP.job_id = B.job_id
ORDER BY C.company_id, JP.job_id;
```

3) Total job postings listed by each company.

```
SELECT C.name AS company_name, COUNT(JP.job_id) AS job_count
FROM companies AS C
LEFT JOIN job_postings AS JP ON C.company_id = JP.company_id
GROUP BY C.name
ORDER BY job_count DESC;
```

4) Job posting with maximum no. of views and total number of applications.

```
SELECT JP.job_id, JS.skill_abr, JP.company_id, C.name, max_views, JP.applies FROM job_postings JP JOIN companies C ON JP.company_id = C.company_id JOIN job_skills JS ON JP.job_id = JS.job_id JOIN ( SELECT MAX(views) AS max_views FROM job_postings ) AS max_views_subquery ON JP.views = max_views_subquery.max_views JOIN ( SELECT job_id, SUM(applies) AS applies FROM job_postings GROUP BY job_id ) AS applies_subquery ON JP.job_id = applies_subquery.job_id;
```

5) Identifying skills in demand.

```
SELECT skill_abr, COUNT(skill_abr) AS skill_count FROM job_skills GROUP BY skill_abr ORDER BY skill_count DESC;
```

6) Company having highest employee count.

```
SELECT EC.company_id, C.name, MAX(EC.employee_count) AS highest_employee_count FROM employee_counts EC INNER JOIN companies AS C ON C.company_id=EC.company_id GROUP BY EC.company_id, C.name ORDER BY highest_employee_count desc LIMIT 1;
```

7) Job postings that are currently active, not expired or closed.

```
SELECT job_id, company_id, title, work_type, pay_period, compensation_type, listed_time, expiry, closed_time, job_posting_url, application_url FROM linkedin.job_postings WHERE job_id NOT IN ( SELECT job_id FROM linkedin.job_postings WHERE closed_time ¡ CURRENT_TIMESTAMP() OR expiry ¡ CURRENT_TIMESTAMP() );
```

8) Location wise top 10 companies with the most job postings

```
SELECT j.company_id, c.name AS company_name, COUNT(*) AS job_posting_count, location, j.title, j.med_salary FROM job_postings AS j JOIN companies AS c ON j.company_id = c.company_id where j.med_salary != ”NULL” GROUP BY j.company_id, company_name, j.title, location, j.med_salary ORDER BY COUNT(*) DESC LIMIT 10;
```

9) Analysis of Job Postings with on Location, Company, Title, and Application Duration and filtered through

Inferred through text by linkedin

```
SELECT j.job_id,j.location, c.name, j.title,
datediff(expiry,listed_time) as Days_to_apply FROM
job_postings AS j JOIN companies AS c ON
j.company_id = c.company_id JOIN benefits AS
jben ON j.job_id = jben.job_id where jben.inferred
= 1 ORDER BY j.location,j.closed_time DESC,
j.max_salary DESC;
```

10) Companies and their competitors sharing the same industry

```
SELECT
ci1.company_id AS company_id,
ci2.company_id AS competitor_company_id,
c1.name AS company_name,
c2.name AS competitor_company_name,
ci1.industry AS shared_industry
FROM company_industries AS ci1 JOIN
company_industries AS ci2 ON ci1.industry =
ci2.industry AND ci1.company_id ¡ ci2.company_id
JOIN companies AS c1 ON ci1.company_id
= c1.company_id JOIN companies AS c2 ON
ci2.company_id = c2.company_id;
```

11) Analyze Job Demand Over month

```
SELECT
Month(expiry) AS month, SUM(views) AS total_views,
COUNT(*) AS job_count
FROM job_postings
GROUP BY month
ORDER BY month;
```

12) Partition Method to get the Annual salary on the max_salary specifying range in where clause.

```
SELECT
j.company_id,
j.formatted_work_type,
j.pay_period,
j.max_salary AS max_salary,
max(round(j.max_salary * 12, 3)) OVER (PARTITION
BY
j.company_id) AS Annual_Salary
FROM job_postings j
where (company_id BETWEEN 1500 AND 2500) and
pay_period ='MONTHLY';
```

13) Analyzing the count of different work types such as Full-Time, Part-Time and Contractor Roles based on their pay_period

```
SELECT pay_period,
sum(formatted_work_type = 'Full-time') AS ”Full-Time Roles”,
sum(formatted_work_type = 'Part-time') AS ”Part-Time Roles”,
sum(formatted_work_type = 'Contract') AS ”Contract
Roles”
FROM job_postings
GROUP BY pay_period
ORDER BY pay_period;
```

14) Analyzing the number of Full-Time opportunities offered for Monthly pay_period by each company

```
SELECT
company_id, pay_period,
COUNT(formatted_work_type) AS ”Total Full-Time
Roles”
FROM job_postings
WHERE formatted_work_type = 'Full-time' and
pay_period=
'MONTHLY'
GROUP BY company_id
ORDER BY company_id;
```

15) Display the Total Roles in each roles types by every company.

```
SELECT
company_id,
formatted_work_type, pay_period,
COUNT(*) OVER (PARTITION BY company_id
ORDER BY
pay_period) AS ”Total Roles”
FROM job_postings
WHERE pay_period != 'NA' and company_id != 0
ORDER BY pay_period;
```

16) Partition Method to get the Annual salary on the max_salary specifying range in where clause.

```
SELECT
j.company_id,
j.formatted_work_type,
j.pay_period,
j.max_salary AS max_salary,
max(round(j.max_salary * 12, 3)) OVER (PARTITION
BY
j.company_id) AS Annual_Salary
FROM job_postings j
where (company_id BETWEEN 1500 AND 2500) and
pay_period
='MONTHLY';
```

B. *Stored Procedures*

1) Companies with vision Insurance as benefit

```
DELIMITER //
CREATE PROCEDURE
GetCompanieswithspecificBenefit(BenefitType
VARCHAR(100))
BEGIN
SELECT DISTINCT j.job_id, c.name AS
company_name
FROM job_postings AS j
join companies as c on c.company_id = j.company_id
JOIN benefits AS b ON j.job_id = b.job_id
WHERE b.type = BenefitType;
END
//
DELIMITER ;

call
GetCompanieswithspecificBenefit('Vision insurance');
```

2) Analysing Industry wise employee count

```
DELIMITER //
CREATE PROCEDURE CalculateEmployeeCountByIndustry()
BEGIN
SELECT ci.industry, c.name, ec.employee_count
FROM company_industries AS ci
INNER JOIN companies AS c on ci.company_id
=c.company_id INNER JOIN employee_counts AS ec
ON
ci.company_id = ec.company_id
GROUP BY ci.industry, c.name,ec.employee_count ;
END
//
DELIMITER ;

call CalculateEmployeeCountByIndustry();
```

3) Get JobPostings WithMetrics by CompanySize for given type of work

```
DELIMITER //
CREATE PROCEDURE
GetJobPostingsWithMetricsAndSortByCompanySize
(type_of_work VARCHAR(100))
BEGIN
SELECT jp.job_id,c.name AS company_name, jp.title,
jp.description, jp.pay_period,
jp.work_type,
c.company_size
FROM job_postings AS jp
INNER JOIN companies AS c ON jp.company_id =
c.company_id
where jp.work_type = type_of_work
ORDER BY c.company_size desc;
END
//
DELIMITER ;

call
GetJobPostingsWithMetricsAndSortByCompanySize
('PART_TIME');
```

4) Find all jobs from a particular industry

```
DELIMITER //
CREATE PROCEDURE
GET_JOB_FROM_INDUSTRY(
IN industry_name VARCHAR(30)
)
BEGIN
SELECT jp.job_id,
jp.company_id, jp.title, jp.work_type,
jp.location, jp.expiry, jp.closed_time,
jp.posting_domain,
jp.job_posting_url,
jp.application_url
FROM linkedin.job_postings AS jp
JOIN linkedin.company_industries AS ci
ON jp.company_id = ci.company_id
WHERE ci.industry = industry_name;
END //
DELIMITER ;

CALL GET_JOB_FROM_INDUSTRY('Retail');
```

5) Display companies offering remote work.

```
DELIMITER //
CREATE PROCEDURE GetRemoteJobs()
BEGIN
SELECT DISTINCT C.name AS 'company name',
JP.location,
JP.title, JP.job_id
FROM companies C
INNER JOIN job_postings JP ON C.company_id =
JP.company_id
WHERE JP.remote_allowed = 1;
END
//
DELIMITER ;
```

6) Procedure to insert values to companies and company_specialities table and get the latest count of total specialities with its corresponding company_id.

```
DELIMITER //
CREATE PROCEDURE
ProcessCompanySpecialitiesWithDeduplicateCompanyID(
IN p_company_id BIGINT,
IN p_speciality VARCHAR(256)
)
BEGIN
INSERT INTO company_specialities (company_id,
speciality)
VALUES (p_company_id, p_speciality)
```

```
ON DUPLICATE KEY UPDATE speciality =
p_speciality;
SELECT company_id, COUNT(*) AS total_specialities
FROM company_specialities
WHERE company_id = p_company_id
GROUP BY company_id;
END //
DELIMITER ;
call
ProcessCompanySpecialitiesWithDeduplicateCompanyID
(999999999,’ZZ
Z 22 TEST Speciality’);
```

7) Procedure to get the exact user desired count as total_specialities_param to produce the company_id, speciality and total_specialities

```
DELIMITER //
CREATE PROCEDURE
GetTotalSpecialities(total_specialities_param int)
BEGIN
select * from (
SELECT *,
COUNT(*) OVER (PARTITION BY company_id) AS
total_specialities
FROM company_specialities
order by (COUNT(*) OVER (PARTITION BY
company_id)) desc
) A
where total_specialities =total_specialities_param;
END
//
DELIMITER ;

call GetTotalSpecialities(20);
```

C. *Triggers*

1) Inserting value in Company activity log table after a deletion of company in companies table

```
DELIMITER //
CREATE TRIGGER Delete_job_posting_with_company
AFTER DELETE ON Companies FOR EACH ROW
BEGIN
UPDATE job_postings SET Company_id = NULL
WHERE
company_id
= old.company_id;
INSERT INTO Company_Activity_logs VALUES (
old.Company_id,
CONCAT(’Row has been deleted from
Job_postings for company
’, old.Company_id));
END //
DELIMITER ;
```

2) Creating a trigger to log changes to the job_postings table

```
CREATE TABLE job_postings_audit (
audit_id INT AUTO_INCREMENT PRIMARY KEY,
job_id INT,
updated_column VARCHAR(50),
old_value VARCHAR(50),
new_value VARCHAR(50) );
```

Create the trigger to log changes to the job_postings table

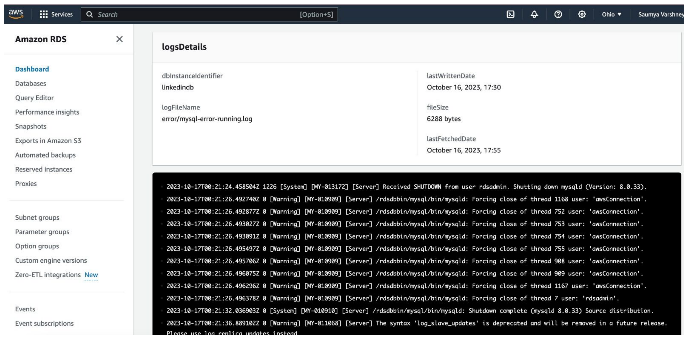
```
DELIMITER //
CREATE TRIGGER JobPostingsAuditTrigger
AFTER UPDATE ON job_postings FOR EACH ROW
BEGIN
INSERT INTO job_postings_audit (job_id,
updated_column,
old_value, new_value)
VALUES (OLD.job_id, ’title’, OLD.title, NEW.title);
INSERT INTO job_postings_audit (job_id,
updated_column,
old_value, new_value)
VALUES (OLD.job_id, ’work_type’, OLD.work_type,
NEW.work_type);
INSERT INTO job_postings_audit (job_id,
updated_column,
old_value, new_value)
VALUES (OLD.job_id, ’location’, OLD.location,
NEW.location);
END
//
DELIMITER ;

UPDATE job_postings
SET title = ’Sales Manager’, work_type
=’FULL_TIME’, location
= ’Santa Clarita’
WHERE job_id = 133114754;
SELECT * FROM job_postings_audit;
```

D. *Access Privileges*

- CREATE USER ’TestUser1’@’localhost’ IDENTIFIED BY ’Test123’; GRANT ALL PRIVILEGES ON linkedin.* TO ’TestUser1’@’localhost’ WITH GRANT OPTION;
- CREATE USER ’TestUser2’@’localhost’ IDENTIFIED BY ’Test1234’; GRANT ALL PRIVILEGES ON linkedin.* TO ’TestUser2’@’localhost’ WITH GRANT OPTION;

VIII. Connection To AWS RDS Using PYTHON

```
import pymysql db = pymysql.connect(host=’linkedindb.
```



logsDetails

dbInstanceIdentifier
linkedindb

logFileName
error/mysql-error-running.log

lastWrittenDate
October 16, 2023, 17:30

fileSize
6288 bytes

lastFetchedDate
October 16, 2023, 17:55

```
2023-10-17T00:21:24.458504Z 1226 [System] [MY-013172] [Server] Received SHUTDOWN from user rdsadmin. Shutting down mysqld (Version: 8.0.33).
2023-10-17T00:21:26.492740Z 0 [Warning] [MY-010909] [Server] /rdsdbbin/mysql/bin/mysqld: Forcing close of thread 1168 user: 'awsConnection'.
2023-10-17T00:21:26.492877Z 0 [Warning] [MY-010909] [Server] /rdsdbbin/mysql/bin/mysqld: Forcing close of thread 752 user: 'awsConnection'.
2023-10-17T00:21:26.493027Z 0 [Warning] [MY-010909] [Server] /rdsdbbin/mysql/bin/mysqld: Forcing close of thread 753 user: 'awsConnection'.
2023-10-17T00:21:26.493091Z 0 [Warning] [MY-010909] [Server] /rdsdbbin/mysql/bin/mysqld: Forcing close of thread 754 user: 'awsConnection'.
2023-10-17T00:21:26.495497Z 0 [Warning] [MY-010909] [Server] /rdsdbbin/mysql/bin/mysqld: Forcing close of thread 755 user: 'awsConnection'.
2023-10-17T00:21:26.495706Z 0 [Warning] [MY-010909] [Server] /rdsdbbin/mysql/bin/mysqld: Forcing close of thread 908 user: 'awsConnection'.
2023-10-17T00:21:26.496075Z 0 [Warning] [MY-010909] [Server] /rdsdbbin/mysql/bin/mysqld: Forcing close of thread 909 user: 'awsConnection'.
2023-10-17T00:21:26.496296Z 0 [Warning] [MY-010909] [Server] /rdsdbbin/mysql/bin/mysqld: Forcing close of thread 1167 user: 'awsConnection'.
2023-10-17T00:21:26.496378Z 0 [Warning] [MY-010909] [Server] /rdsdbbin/mysql/bin/mysqld: Forcing close of thread 7 user: 'rdsadmin'.
2023-10-17T00:21:32.036903Z 0 [System] [MY-010910] [Server] /rdsdbbin/mysql/bin/mysqld: Shutdown complete (mysqld 8.0.33) Source distribution.
2023-10-17T00:21:36.889102Z 0 [Warning] [MY-011068] [Server] The syntax 'log_slave_updates' is deprecated and will be removed in a future release.
Please use log_replica_updates instead
```

Fig. 3. Screenshot for Logging of database

clcslgwnrrnu.us-east-2.rds.amazonaws.com',
user='awsConnection',password='qwerty1234')
cursor = db.cursor()
cursor.execute('select version()')
data=cursor.fetchone()
sql = "'use linkedin"'
cursor.execute(sql)
sql = "'show tables"'
cursor.execute(sql)
cursor.fetchall()

This module is Identifying company with highest number of employee.

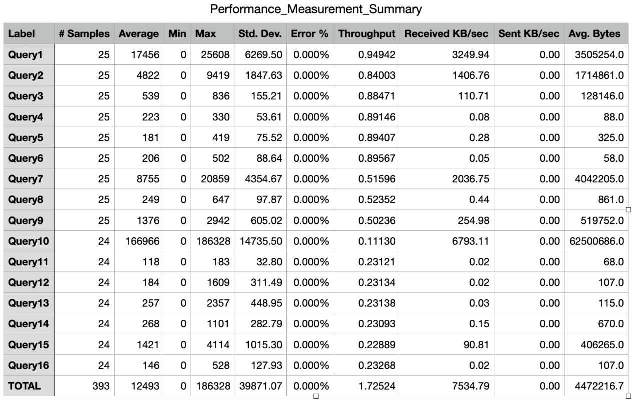
sql = "'SELECT EC.company_Id, C.name,
MAX(EC.employee_count) AS Highest_Employee_count
FROM employee_counts EC
INNER JOIN companies AS C
ON C.company_id=EC.company_id
GROUP BY EC.company_id, C.name
ORDER BY highest_employee_count desc
LIMIT 1; "'
cursor.execute(sql)
result = cursor.fetchall()
for i in result:
print(" has the highest employee count with
employees.".format(i[1], i[2]))
cursor.close()
db.close()

IX. LOGGING OF DB

AWS connectivity is the biggest advantage for monitoring our logs. Accessing the SQL workbench through AWS connection will generate a log which is captured by Amazon CloudWatch service. Below screenshots capture that information about the queries when run through the AWS-connection with MySQL workbench. (Refer Fig.3)

X. SQL PERFORMANCE MEASUREMENT

Apache JMeter, the performance measurement tool which will help to measure the performance of the SQL queries using multiple threads. On establishing the connection of the MySQL workbench with the JMeter, here, the performances of queries were analyzed using 25 threads. The reports are generated, and the screenshot of the summary report is displayed below.



Performance_Measurement_Summary

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
Query1	25	17456	0	25608	6269.50	0.000%	0.94942	3249.94	0.00	3505254.0
Query2	25	4822	0	9419	1847.63	0.000%	0.84003	1406.76	0.00	1714861.0
Query3	25	539	0	836	155.21	0.000%	0.88471	110.71	0.00	128146.0
Query4	25	223	0	330	53.61	0.000%	0.89146	0.08	0.00	88.0
Query5	25	181	0	419	75.52	0.000%	0.89407	0.28	0.00	325.0
Query6	25	206	0	502	88.64	0.000%	0.89567	0.05	0.00	58.0
Query7	25	8755	0	20859	4354.67	0.000%	0.51596	2036.75	0.00	4042205.0
Query8	25	249	0	647	97.87	0.000%	0.52352	0.44	0.00	861.0
Query9	25	1376	0	2942	605.02	0.000%	0.50236	254.98	0.00	519752.0
Query10	24	166966	0	186328	14735.50	0.000%	0.11130	6793.11	0.00	62500686.0
Query11	24	118	0	183	32.80	0.000%	0.23121	0.02	0.00	68.0
Query12	24	184	0	1609	311.49	0.000%	0.23134	0.02	0.00	107.0
Query13	24	257	0	2357	448.95	0.000%	0.23138	0.03	0.00	115.0
Query14	24	268	0	1101	282.79	0.000%	0.23093	0.15	0.00	670.0
Query15	24	1421	0	4114	1015.30	0.000%	0.22889	90.81	0.00	406265.0
Query16	24	146	0	528	127.93	0.000%	0.23268	0.02	0.00	107.0
TOTAL	393	12493	0	186328	39871.07	0.000%	1.72524	7534.79	0.00	4472216.7

Fig. 4. Screenshot for Performance Measurement

XI. GITHUB REPOSITORY

https://github.com/NehaBais/kaggle-linkedin-jobposting-analysis/branches

XII. CONCLUSION

The purpose of the LinkedIn dataset conveyed the essence of a huge business process established between the job seeker and the job recruiter. The dataset helped to come up with various interpretations and drawing possible outcomes along with some limitations. Our functional analysis demonstrated the most accurate rendering of the dataset to a job seeker and a job recruiter.