USE CASE STUDY REPORT : Milestone 2

Group No.: Group 15

Student Names: Vraj Diyora and Amruta Hombali

I. Introduction

The job search process can often be time-consuming for both job seekers and employers. It can be challenging to navigate through various websites and identify the best opportunities. To address this problem, our team has developed a Job Search Tool Cross-Platform that scours through multiple job sites and consolidates all job listings in one place. This platform is designed to simplify the job search process for both job seekers and employers by providing a comprehensive database of job listings from different sources.

The purpose of this report is to provide insights into the Job Search Tool Cross Platform database, specifically through the use of SQL queries to analyze job listings, employers, and job seekers. By doing so, we aim to gain a better understanding of the job market and provide valuable information for both job seekers and employers. Specifically, we aim to answer questions about the distribution of job postings by location, the companies that are actively hiring, the most popular job listings, and the salaries offered for different job titles and locations.

To conduct this study, we used a relational database schema with tables for job listings, employers, and job seekers. The schema was populated with realistic sample data to provide a representative snapshot of the job market. We used SQL and NoSQL queries to extract and analyze the data, and visualizations were created using Python libraries such as Matplotlib.

Overall, this study provides valuable insights into the job market and the use of SQL queries to analyze job listings, employers, and job seekers. The findings can be used by both job seekers and employers to make more informed decisions and improve their chances of finding the perfect match.

II. Conceptual Data Modeling

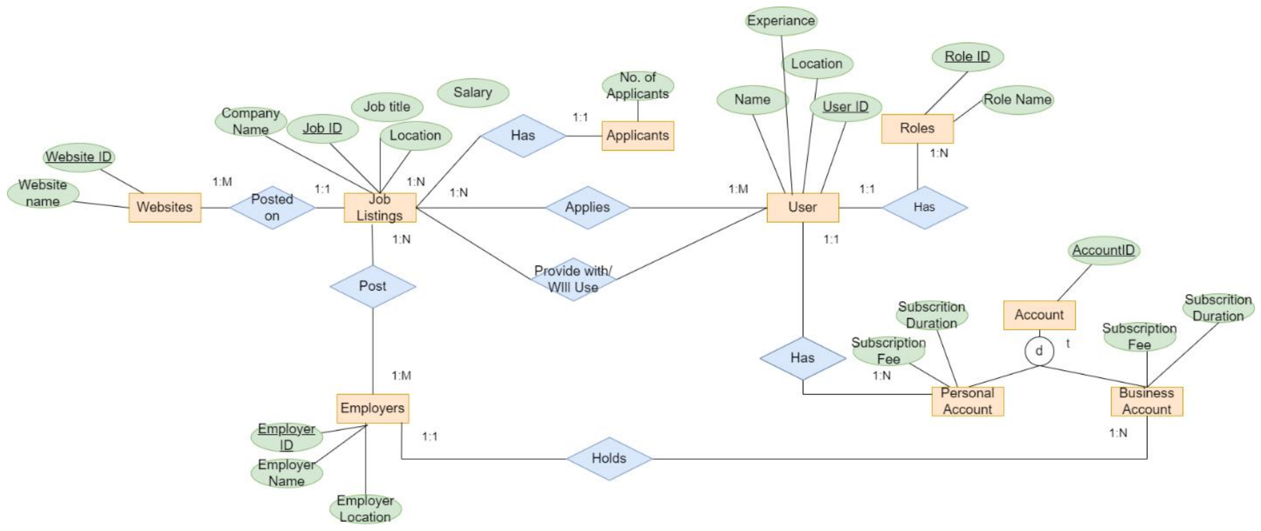
The first step in designing a database is to create a data model that represents the entities and relationships of the domain being modeled. In this project, we used two modeling techniques: EER (Enhanced Entity-Relationship) and UML (Unified Modeling Language).

In our EER model, we defined entities such Websites, Job Listings, User/Jobseeker, Employers, Applicants, Account, and Roles. Each having specific attributes. For ex :The Job entity has attributes such as JobID, Title, Location, and Salary.These entities have relationships with each other, such as Job being posted by a Company, a Job having multiple Applicants, and so on.

The cardinalities of the relationships between entities are also important to consider. For example, a User can apply to multiple Job Listings, and a Job Listing can have multiple Applicants. This is represented as a many-to-many relationship between the User and Job Listing entities. Similarly, a website can post multiple Job Listings, and a Job Listing can be posted by only one website. This is represented as a one-to-many relationship between the Website and Job Listing entities.

Similarly the UML diagram was created to include the different classes.

EER Model



III. **Mapping the Relational Model based on the conceptual data model:**

After developing the conceptual data model using EER and UML class diagrams, the next step is to map it to a relational model. The relational model consists of tables with columns representing attributes and relationships between tables.

To map our conceptual data model to a relational model, we started by identifying the main entities and relationships in our model. We then created tables for each entity, using the entity's attributes as columns in the table. We also created tables for each relationship, using the foreign keys of the related entities as columns in the table.

Our relational model consists of 11 tables, each representing a different entity or relationship in our conceptual data model. These tables are: Job Listing, Website, Applicants, Employer, Job Posts, User, AppliesforJob, Role, Account, Personal Account, and Business Account.

We have identified the primary key for each table, which is used to uniquely identify each record in the table. We have also identified foreign keys for each table, which are used to create relationships between tables. For example, the JobID field in the Website table is a foreign key that references the JobID field in the Job Listing table.

To ensure that our relational model is in the first normal form (1NF), we have ensured that each table has a primary key and each attribute has a single value, and each column has atomic values.

There are no repeating groups in any column. We achieved this by creating a table for each relationship and ensuring that every attribute has a single value.

To further normalize our relational model, we applied the second normal form (2NF) by ensuring that all non-key attributes in each table depend on the primary key. For example, we separated the WebsiteName attribute from the Job Listing table and created a separate Website table, as the WebsiteName attribute only depends on the WebsiteID foreign key. Similarly, we separated the SubscriptionFee and SubscriptionDuration attributes from the Employer table and created a separate Business Account table, as these attributes only depend on the BusinessActID foreign key.

We also separated the RoleName attribute from the User table and created a separate Role table, as the RoleName attribute only depends on the RoleID foreign key. Finally, we separated the PersonalAcctID attribute from the User table and created a separate Personal Account table, as the SubscriptionFee and SubscriptionDuration attributes only depend on the PersonalAcctID foreign key.

Overall, our relational model effectively maps our conceptual data model to a set of normalized tables, with clearly defined relationships between entities and attributes.

Relational Model :

Job Listing (JobID, Company Name, Job Title, Location, Salary)

Website (WebsiteID**,** *JobID***,** WebsiteName**)**
Foreign Key – *JobID* refers to JobID in Job listing. NULL not allowed.

Applicants (*JobID***,** Number of Applicants**)**
Foreign Key – *JobID* refers to JobID in Job listing. NULL not allowed.

Employer (EmployerID**,** Employer Name, Employer Location**)**

Job Posts (*JobID, EmployerID***)**
Foreign Key – *JobID* refers to JobID in Job listing. NULL not allowed.
Foreign Key – *EmployerID* refers to EmployerID in Employer. NULL not allowed.

User (UserID**,** UserName, UserLocation, UserExperiance**)**

AppliesforJob (*JobID, UserID***)**
Foreign Key – *JobID* refers to JobID in Job listing. NULL not allowed.
Foreign Key – *UserID* refers to UserID in User. NULL not allowed.

Role (RoleID**,** *UserID*, Role Name**)**
Foreign Key – *UserID* refers to UserID in User . NULL not allowed.

Account (AccountID**)**

Personal Account (PersonalActID, *UserID*, SubscriptionFee, SubscriptionDuration**)**

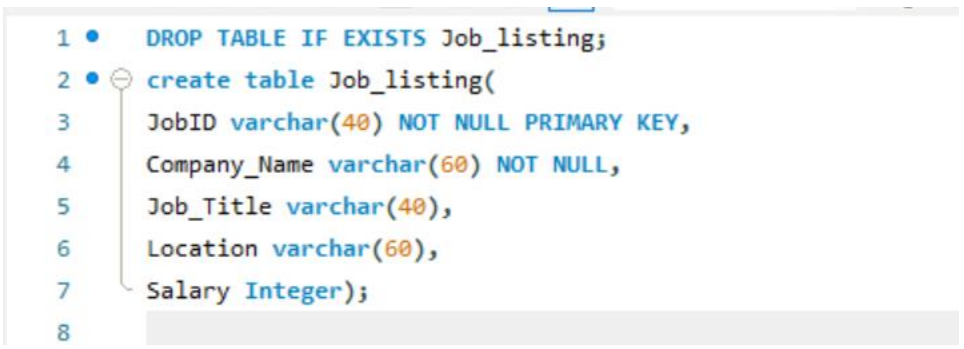
Foreign Key – *UserID* refers to UserID in User. NULL not allowed.

Business Account (<u>BusinessActID,</u> *EmployerID*, SubscriptionFee, SubscriptionDuration**)**
Foreign Key – *EmployerID* refers to EmployerID in Employer. NULL not allowed.

IV. Implementation of Relation Model via MySQL and NoSQL

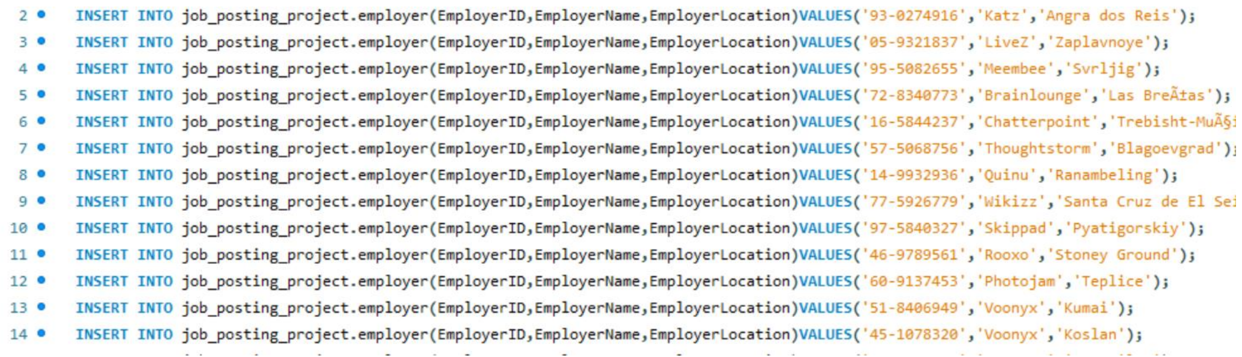
In this section, we will discuss the implementation of the relational model for the Job Search tool cross-platform database using MySQL and NoSQL.

To implement the relational model, we created the necessary tables in SQL based on the schema derived from the conceptual data model. We used MySQL as our relational database management system for this project. To create the tables, we used the CREATE TABLE statement using Data Defition Language (DDL) in SQL, specifying the name of the table, the column names, and the data types for each column. We also included any necessary constraints, such as primary keys and foreign keys.



```
DROP TABLE IF EXISTS Job_listing;
create table Job_listing(
JobID varchar(40) NOT NULL PRIMARY KEY,
Company_Name varchar(60) NOT NULL,
Job_Title varchar(40),
Location varchar(60),
Salary Integer);
```

After creating the tables, we populated them with realistic sample data. To do this, we used SQL INSERT statements to add records to each table. We ensured that the data was same as the data types and constraints specified in the table definitions.



```
INSERT INTO job_posting_project.employer(EmployerID,EmployerName,EmployerLocation)VALUES('93-0274916','Katz','Angra dos Reis');
INSERT INTO job_posting_project.employer(EmployerID,EmployerName,EmployerLocation)VALUES('05-9321837','LiveZ','Zaplavnoye');
INSERT INTO job_posting_project.employer(EmployerID,EmployerName,EmployerLocation)VALUES('95-5082655','Meembee','Svrljig');
INSERT INTO job_posting_project.employer(EmployerID,EmployerName,EmployerLocation)VALUES('72-8340773','Brainlounge','Las BreÃ±as');
INSERT INTO job_posting_project.employer(EmployerID,EmployerName,EmployerLocation)VALUES('16-5844237','Chatterpoint','Trebisht-MuÃ§i
INSERT INTO job_posting_project.employer(EmployerID,EmployerName,EmployerLocation)VALUES('57-5068756','Thoughtstorm','Blagoevgrad');
INSERT INTO job_posting_project.employer(EmployerID,EmployerName,EmployerLocation)VALUES('14-9932936','Quinu','Ranambeling');
INSERT INTO job_posting_project.employer(EmployerID,EmployerName,EmployerLocation)VALUES('77-5926779','Wikizz','Santa Cruz de El Sei
INSERT INTO job_posting_project.employer(EmployerID,EmployerName,EmployerLocation)VALUES('97-5840327','Skippad','Pyatigorskiy');
INSERT INTO job_posting_project.employer(EmployerID,EmployerName,EmployerLocation)VALUES('46-9789561','Rooxo','Stoney Ground');
INSERT INTO job_posting_project.employer(EmployerID,EmployerName,EmployerLocation)VALUES('60-9137453','Photojam','Teplice');
INSERT INTO job_posting_project.employer(EmployerID,EmployerName,EmployerLocation)VALUES('51-8406949','Voonyx','Kumai');
INSERT INTO job_posting_project.employer(EmployerID,EmployerName,EmployerLocation)VALUES('45-1078320','Voonyx','Koslan');
```

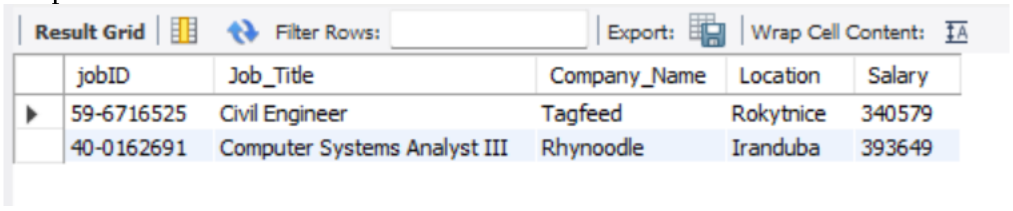
To derive valuable business insights, we have provided SQL queries for some common scenarios that the Analytics team at Job Search tool cross platform may encounter. Some of them are :

- The list of jobs which includes the JobId, Job Title, location, Salary that were posted on a particular website called 'My works'.

  Select j.jobID, j.Job_Title, j.Company_Name, j.Location, j.Salary
  From job_listing j
  Inner Join website w on j.JobID=w.JobID

Where websitename = 'myworks';

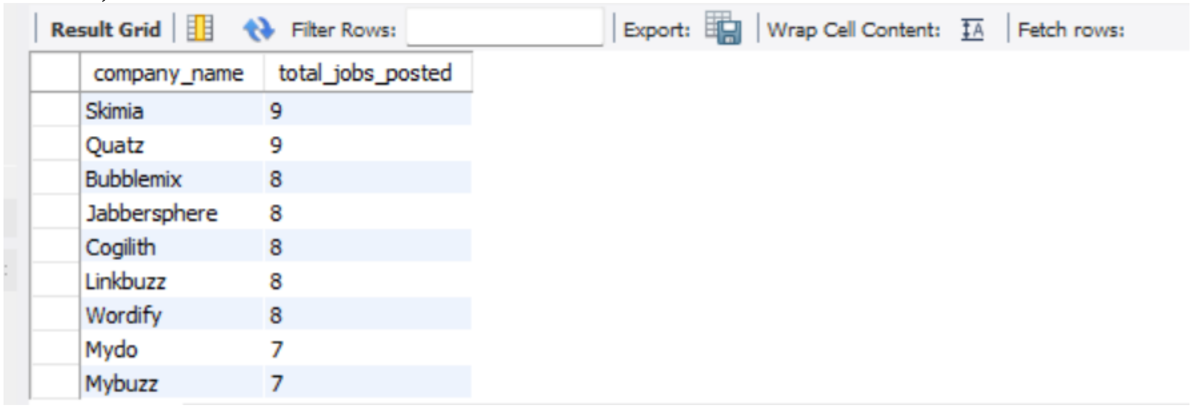
Output:



	jobID	Job_Title	Company_Name	Location	Salary
▶	59-6716525	Civil Engineer	Tagfeed	Rokytnice	340579
	40-0162691	Computer Systems Analyst III	Rhynoodle	Iranduba	393649

- The total number of applicants that applied for a job title named ‘Paralegal’.
- The list of companies that are actively hiring and posting multiple projects on the platform, particularly companies that posted more than 5 jobs in descending order.

Select company_name, **count(*)** as total_jobs_posted
From job_listing
Group by company_name
Having count(*)>5
Order by count(*) desc
Limit 10;

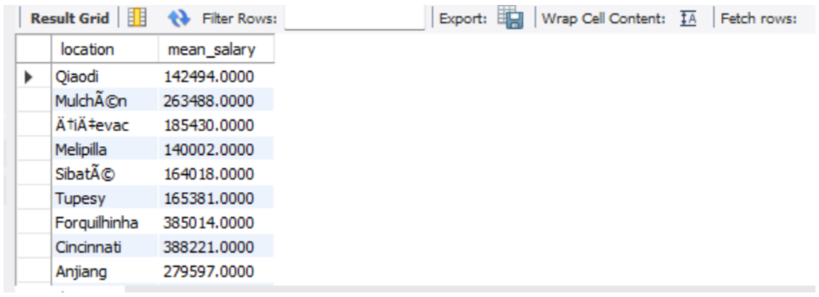


company_name	total_jobs_posted
Skimia	9
Quatz	9
Bubblemix	8
Jabbersphere	8
Cogilith	8
Linkbuzz	8
Wordify	8
Mydo	7
Mybuzz	7

- Average Salary offered in various locations.

Select location, **avg**(salary) as mean_salary
From job_listing
Group by location
Having avg(salary)> 120000
Limit 10;

<u>Output:</u>

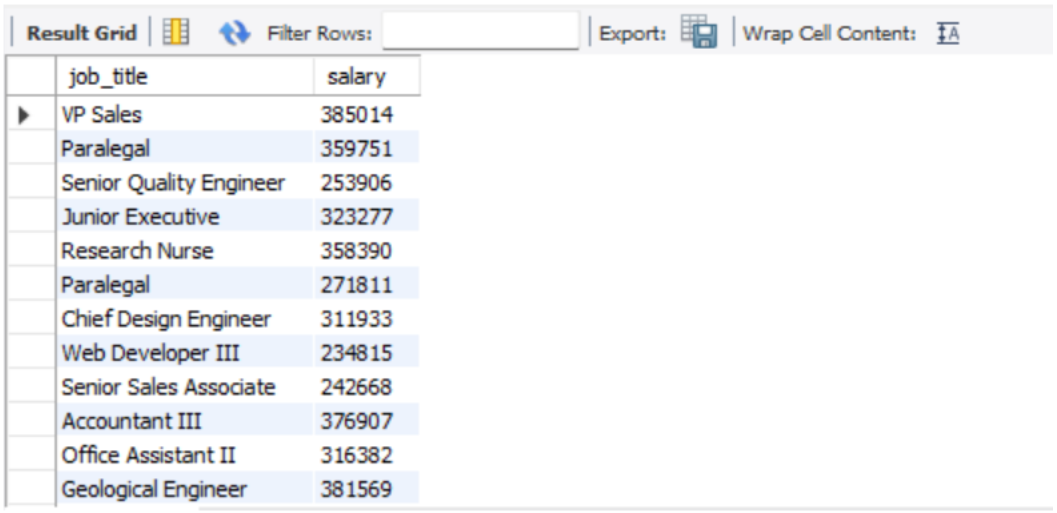


	location	mean_salary
▶	Qiaodi	142494.0000
	MulchÃ©n	263488.0000
	Ä†iÄ‡evac	185430.0000
	Melipilla	140002.0000
	SibatÃ©	164018.0000
	Tupesy	165381.0000
	Forquilhinha	385014.0000
	Cincinnati	388221.0000
	Anjiang	279597.0000

- The top 10 most popular job listings (by number of applicants)
- The Job Titles that are most in demand interms of location and compensation offered.
- The companies offering higher salaries than their avg salary for that job title

```
SELECT job_title, salary
FROM job_listing j
WHERE salary > (
SELECT AVG(salary)
FROM job_listing
WHERE job_title = j.job_title
);
```

<u>Output:</u>

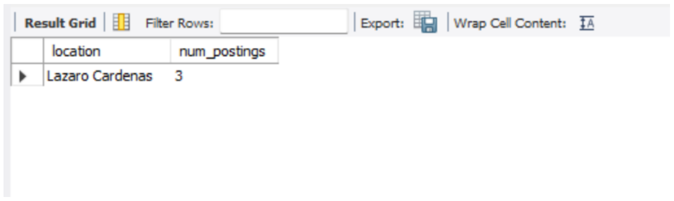


job_title	salary
VP Sales	385014
Paralegal	359751
Senior Quality Engineer	253906
Junior Executive	323277
Research Nurse	358390
Paralegal	271811
Chief Design Engineer	311933
Web Developer III	234815
Senior Sales Associate	242668
Accountant III	376907
Office Assistant II	316382
Geological Engineer	381569

- The location having the highest demand for jobs

```
WITH temp_table AS
(SELECT location
,COUNT(*) as num_postings
FROM job_listing
GROUP BY location
)
SELECT location
,COUNT(*) as num_postings
FROM job_listing
GROUP BY location
HAVING COUNT(*) = (SELECT MAX(num_postings) FROM temp_table);
```

<u>Output:</u>



location	num_postings
Lazaro Cardenas	3

These queries demonstrate the usefulness of the database for deriving business insights and analyzing job search trends. By using the data stored in this database, the Analytics team can make data decisions to improve the platform and attract more job seekers and employers.

In addition to Mysql, we also explored NoSQL databases like MongoDB for storing our data into a database. We defined collections for each entity by running the following commands: db.createCollection("job_listings"). We inserted the values using the following command : db.applicants.insertOne({ JobID: "52-6528690", NumberOfApplicants: 10 }). Here were some of the business problems that we queried were :

- The jobs that are most popular and having the most number of applicants

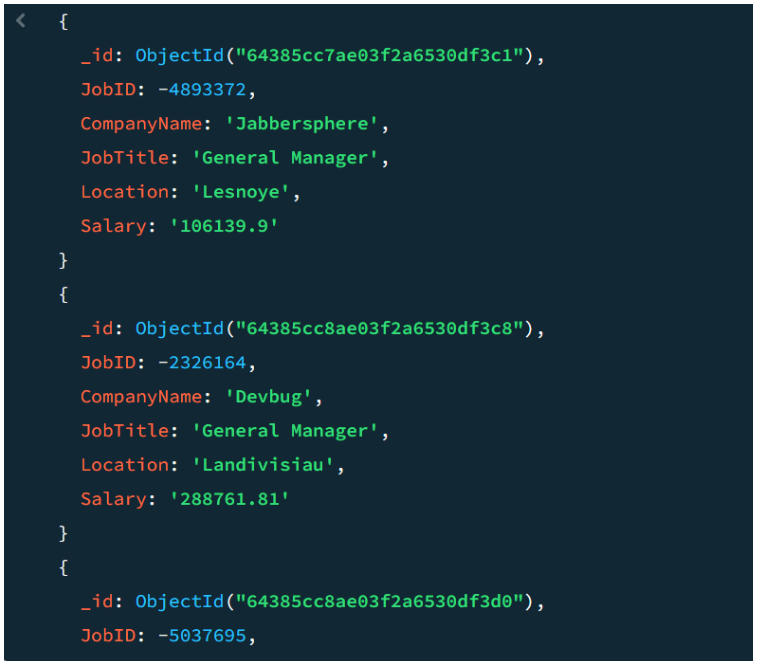
  db.applicants.find().sort({NumberofApplicants: -1}).limit(10)

- The jobs that are paying the highest salaries

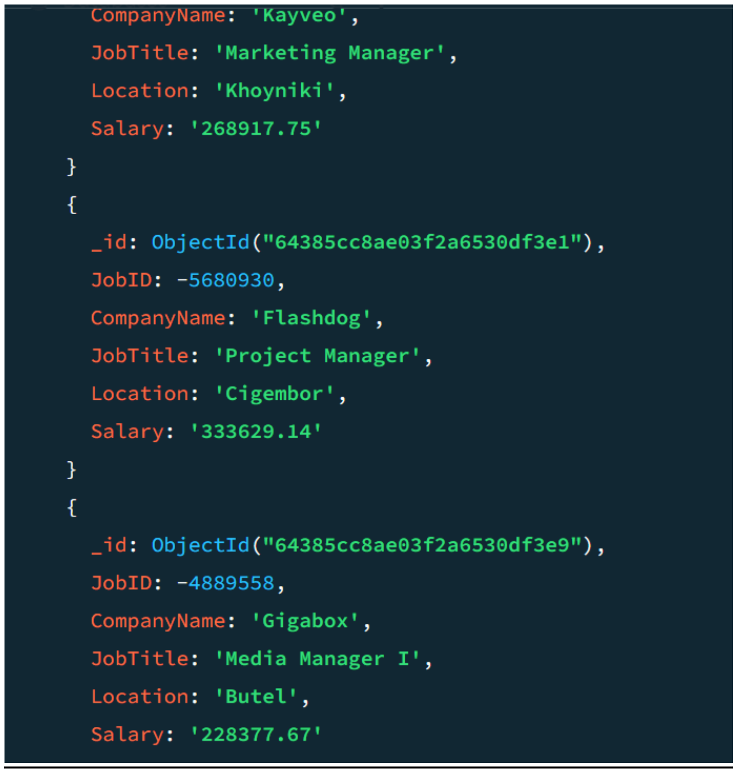
  db.job_listings.find({}, { _id: 0, JobID: 1, CompanyName: 1, Salary: 1 })
  .sort({ Salary: -1 })
  .limit(10)

- All job listings with a job title that contains the word "Manager"

  db.job_listings.find({ JobTitle: /Manager/ })



```
<  {
     _id: ObjectId("64385cc7ae03f2a6530df3c1"),
     JobID: -4893372,
     CompanyName: 'Jabbersphere',
     JobTitle: 'General Manager',
     Location: 'Lesnoye',
     Salary: '106139.9'
   }
   {
     _id: ObjectId("64385cc8ae03f2a6530df3c8"),
     JobID: -2326164,
     CompanyName: 'Devbug',
     JobTitle: 'General Manager',
     Location: 'Landivisiau',
     Salary: '288761.81'
   }
   {
     _id: ObjectId("64385cc8ae03f2a6530df3d0"),
     JobID: -5037695,
```



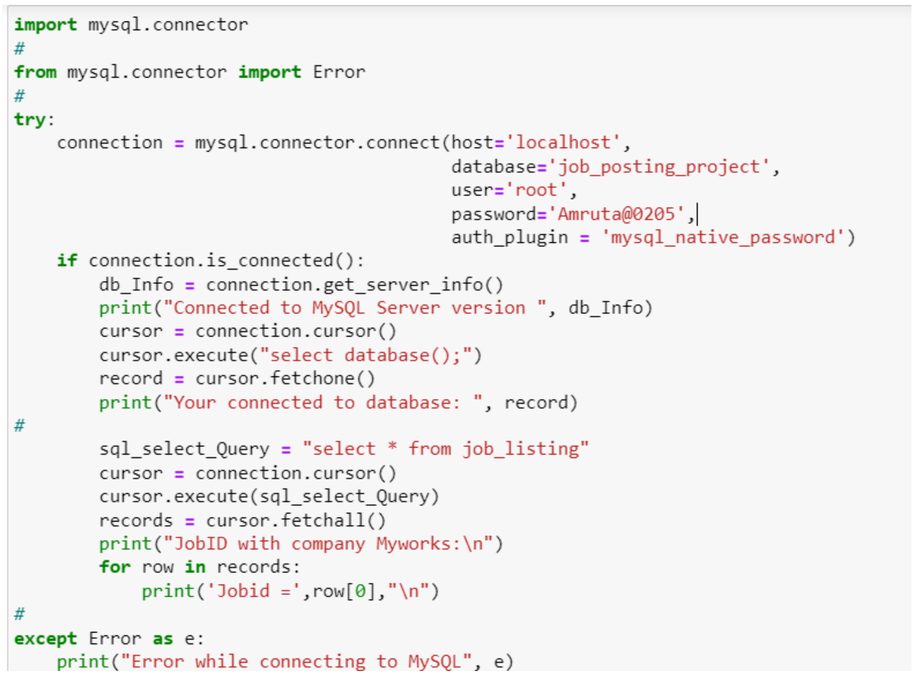
```
    CompanyName: 'Kayveo',
    JobTitle: 'Marketing Manager',
    Location: 'Khoyniki',
    Salary: '268917.75'
  }
  {
    _id: ObjectId("64385cc8ae03f2a6530df3e1"),
    JobID: -5680930,
    CompanyName: 'Flashdog',
    JobTitle: 'Project Manager',
    Location: 'Cigembor',
    Salary: '333629.14'
  }
  {
    _id: ObjectId("64385cc8ae03f2a6530df3e9"),
    JobID: -4889558,
    CompanyName: 'Gigabox',
    JobTitle: 'Media Manager I',
    Location: 'Butel',
    Salary: '228377.67'
```

Overall, the implementation of the relational model via MySQL and NoSQL for the Job Search tool cross-platform database provides a robust and scalable solution for managing job listings data.

V. Database Access via Python

In order to access the data stored in the MySQL database, we used the mysql.connector package in Python. To connect to the database, we first specified the database host, username, and password respectively. We also specified the name of the database we wanted to connect to using the database argument. Once we had this information, we used the mysql.connector.connect() function to establish a connection to the database. Once we had established a connection to the database, we could then use SQL queries to extract the data we needed for further analysis.

After successfully connecting to the MySQL database, we can perform exploratory data analysis using Python to gain further insights into the data. For example, we can use Python's pandas library to query the database and create data frames to analyze the data With the help of pandas, we can filter and manipulate the data based on our requirements, and then visualize it using popular libraries such as matplotlib. By doing so, we can get a better understanding of the job search trends and patterns in the data, which can help us make informed decisions.

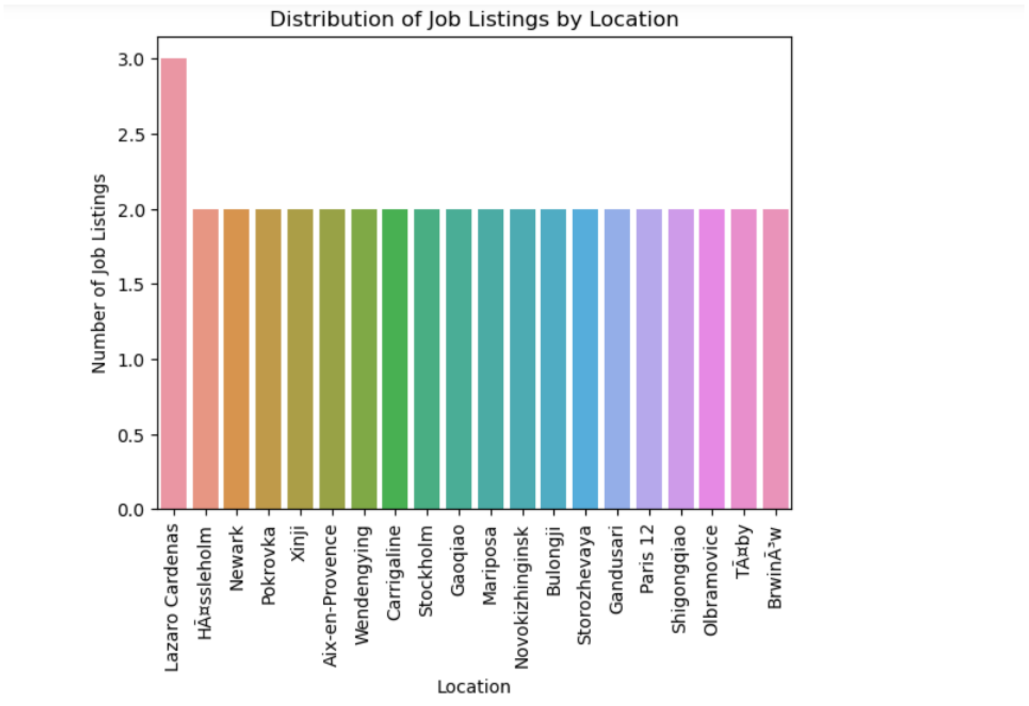


```python
import mysql.connector
#
from mysql.connector import Error
#
try:
    connection = mysql.connector.connect(host='localhost',
                                         database='job_posting_project',
                                         user='root',
                                         password='Amruta@0205',
                                         auth_plugin = 'mysql_native_password')
    if connection.is_connected():
        db_Info = connection.get_server_info()
        print("Connected to MySQL Server version ", db_Info)
        cursor = connection.cursor()
        cursor.execute("select database();")
        record = cursor.fetchone()
        print("Your connected to database: ", record)
#
        sql_select_Query = "select * from job_listing"
        cursor = connection.cursor()
        cursor.execute(sql_select_Query)
        records = cursor.fetchall()
        print("JobID with company Myworks:\n")
        for row in records:
            print('Jobid =',row[0],"\n")
#
except Error as e:
    print("Error while connecting to MySQL", e)
```

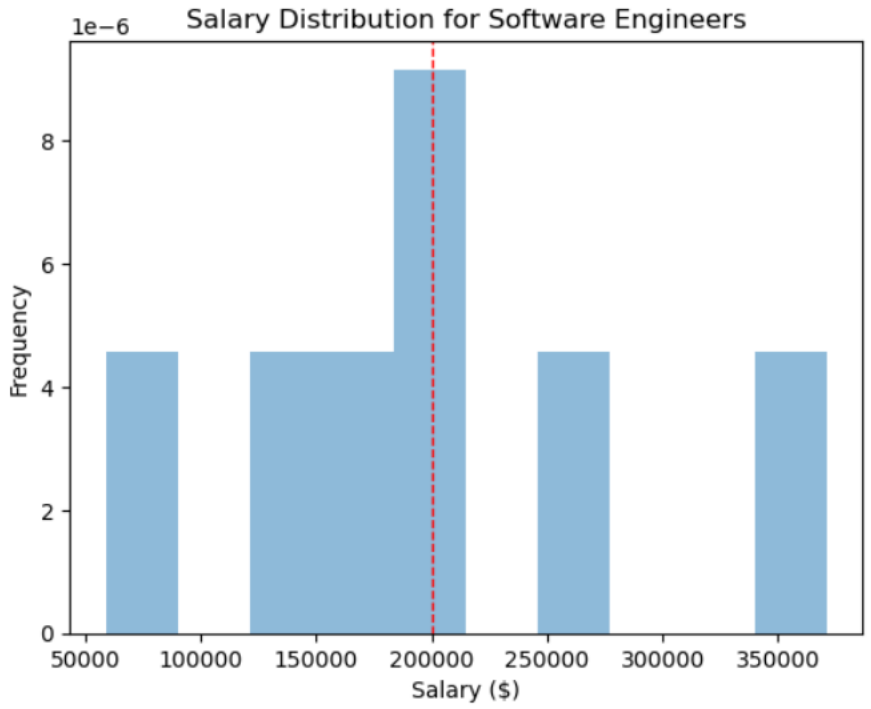
After successfully connecting to the MySQL database, we can perform exploratory data analysis using Python to gain further insights into the data. For example, we can use Python's pandas library to query the database.

Some of the exploratory data analysis that we did answered some of the following business questions :

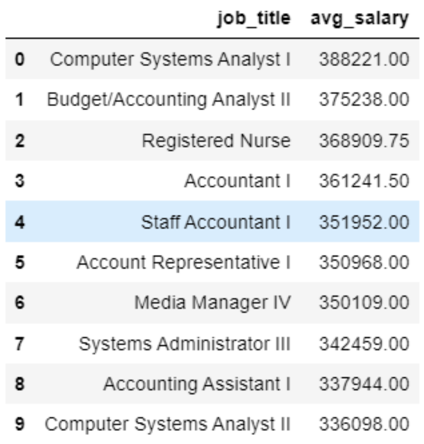
1) What is the distribution of job listings by location?



2) Salary distribution for software engineers :

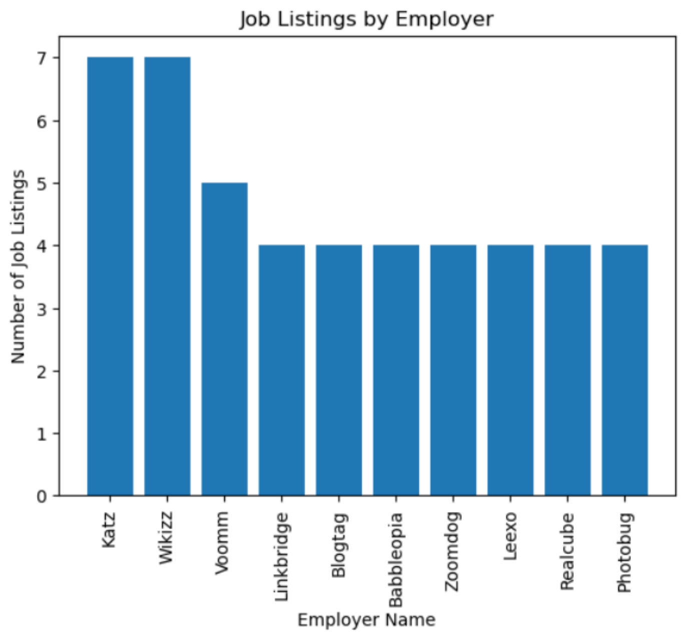


3) What is the distribution of job listings by location?



	job_title	avg_salary
0	Computer Systems Analyst I	388221.00
1	Budget/Accounting Analyst II	375238.00
2	Registered Nurse	368909.75
3	Accountant I	361241.50
4	Staff Accountant I	351952.00
5	Account Representative I	350968.00
6	Media Manager IV	350109.00
7	Systems Administrator III	342459.00
8	Accounting Assistant I	337944.00
9	Computer Systems Analyst II	336098.00

4) What is the distribution of job listings by Employers?



Overall, our exploratory data analysis provided valuable insights into the job search tool cross platform database. By connecting Python to the MySQL database and using SQL queries, we were able to extract and analyze data to answer specific questions and gain a deeper understanding of the job market.

VII. Summary and recommendation

In this use case study, we developed a database for a job search tool cross platform using a relational data model. We created the schema based on the conceptual data model and populated the tables with realistic sample data. We also provided SQL queries to demonstrate the usefulness of the database for generating insights and analyzing job search trends. Additionally, we accessed the database through Python to perform exploratory data analysis and answer some additional questions.

One of the advantages of using a relational database for a job search tool cross platform is that it allows for efficient storage and retrieval of large amounts of data. With the help of SQL, we can easily manipulate and analyze data to derive business insights. Also, connecting the database to Python allows for more advanced analysis and data visualization.

However, there are some limitations and suggestions are provided for them as below:

- There is no information regarding the date that the job was posted, we could try to include this data so as to better filter it for job seekers.
- If a joblisting has multiple locations then it cannot be accommodated in our database, we could include another attribute for futher expansion of project.
- The sample data that we used may not be representative of the real-world job listings, hence we could use realistic data to get more updated and relevant information.

- The current database design does not account for user profiles or preferences, which could limit the ability to provide personalized job recommendations or search results.
- As the size of the database grows, performance issues may arise when executing complex queries or generating report, measures like query optimization needs to be undertaken to limit this.