JOB SEARCH TOOL WITH SQL

- **Project Overview:**
  - **Description:** Designed and implemented a job search tool to enhance the job search experience for users by providing advanced filtering and data management.
  - **Tools Used**: SQL, database management system MySQL**.**

- **Project Steps: Database Structure and Schema Design**

  - **Entities**: Built an optimized schema with tables representing the core entities in a job search platform:
    - ✓ **Jobs**: Stores information about job listings, including job title, location, salary range, date posted, and company ID.
    - ✓ **Companies**: Contains details about the company's offering jobs, such as company ID, company name, industry, and location.
    - ✓ **Job_Seekers**: Holds data about job seekers, including seeker name, location, and education.
    - ✓ **Applications**: Represents job applications, linking job seekers to specific job postings with fields like application date and job ID.
  - **Relationships**: Defined foreign key relationships to establish connections among tables, such as linking **Jobs** to **Companies**, and **Applications** to both **Jobs** and **Job_Seekers**. This relational schema design enabled efficient data retrieval and cross-table analysis.
- **Data Import**:
  - Imported and cleaned the provided job dataset using SQL, ensuring accurate data types and consistent formatting.
  - Executed data transformation queries to make the data analysis ready.
- **SQL Queries:**

BEGINNER-LEVEL

1. **Find all job titles and their locations.**

```
SELECT title, location

FROM jobs;
```

✓ The query selects job titles and their locations from the `jobs` table, providing a quick overview of where each job is based.

2. **List all job seekers located in New York.**

```
SELECT name

FROM job_seekers

WHERE location = 'New York';
```

✓ This query retrieves the names of job seekers located specifically in New York, filtering based on location in the `job_seekers` table.

3. Retrieve the count of applications for each job.

```
SELECT job_id, COUNT(application_id) AS application_count

FROM applications

GROUP BY job_id;
```

✓ This query groups applications by `job_id` and counts the number of applications per job, showing which jobs are most popular.

4. List all unique job locations.

```
SELECT DISTINCT location

FROM jobs;
```

✓ By selecting distinct values from the `location` column in the `jobs` table, this query provides a unique list of all job locations available.

5. Find the names and education levels of job seekers with a "B.Sc. Computer Science" degree.

```
SELECT name, education

FROM job_seekers

WHERE education = 'B.Sc. Computer Science';
```

✓ The query filters job seekers by their education to identify those with a "B.Sc. Computer Science" degree, listing their names and education levels.

INTERMEDIATE-LEVEL

6. List job titles and the names of the companies offering them.

```
SELECT j.title, c.name AS company_name

FROM jobs j

JOIN companies c ON j.company_id = c.company_id;
```

✓ Using a join between `jobs` and `companies`, this query lists each job title alongside the name of the company offering it.

7. Count the number of job seekers in each location.

```
SELECT location, COUNT(seeker_id) AS num_seekers

FROM job_seekers

GROUP BY location;
```

- ✓ This query groups job seekers by location and counts them, allowing insight into how many job seekers reside in each area.

8. Find jobs with a salary range that starts from at least $80,000.

```
SELECT title, salary_range

FROM jobs

WHERE CAST(SUBSTRING_INDEX(salary_range, '-', 1) AS UNSIGNED) >= 80000;
```

- ✓ By casting the start of the `salary_range` as an integer and setting a threshold, this query filters for jobs with a starting salary of at least $80,000.

9. List the top 3 job seekers who have applied to the most jobs.

```
SELECT seeker_id, COUNT(application_id) AS num_applications

FROM applications

GROUP BY seeker_id

ORDER BY num_applications DESC

LIMIT 3;
```

- ✓ The query ranks job seekers based on the number of applications submitted, returning the top three who applied to the most jobs.

10. Retrieve all jobs posted in the last 30 days.

```
SELECT title, date_posted

FROM jobs

WHERE date_posted >= CURDATE() - INTERVAL 30 DAY;
```

- ✓ Using a date filter, this query identifies jobs posted in the last 30 days by comparing `date_posted` to the current date minus 30 days.

11. Find the average salary range for each company.

```
SELECT c.name AS company_name, AVG(CAST(SUBSTRING_INDEX(salary_range, '-', -
1) AS UNSIGNED)) AS avg_salary

FROM jobs j

JOIN companies c ON j.company_id = c.company_id

GROUP BY c.name;
```

- ✓ The query joins `jobs` and `companies` to calculate the average salary for each company, aggregating salary data by company.

12. List jobs with more than 5 applications.

```
SELECT job_id, COUNT(application_id) AS num_applications

FROM applications
```

```
GROUP BY job_id

HAVING num_applications > 5;
```

✓ By grouping applications by `job_id` and using a `HAVING` clause, this query identifies jobs that have received more than five applications.

13. Find the number of applications for each company.

```
SELECT c.name AS company_name, COUNT(a.application_id) AS num_applications

FROM companies c

JOIN jobs j ON c.company_id = j.company_id

JOIN applications a ON j.job_id = a.job_id

GROUP BY c.name;
```

✓ Using a join across `companies`, `jobs`, and `applications`, this query counts applications per company, providing a view of each company's application volume.

14. Retrieve jobs that require a degree in Data Science.

```
SELECT j.title

FROM jobs j

JOIN job_seekers s ON j.location = s.location

WHERE s.education = 'B.A. Data Science';
```

✓ The query joins `jobs` and `job_seekers` and filters for job seekers with a "B.A. Data Science" degree, listing jobs in the same location that match their qualifications.

15. List each job seeker's latest application date.

```
SELECT seeker_id, MAX(application_date) AS latest_application

FROM applications

GROUP BY seeker_id;
```

✓ This query groups applications by `seeker_id` and finds the most recent application date, giving the last date each job seeker applied.

16. Find the job with the highest number of applications.

```
SELECT job_id, COUNT(application_id) AS num_applications

FROM applications

GROUP BY job_id

ORDER BY num_applications DESC

LIMIT 1;
```

✓ By grouping applications by `job_id` and ordering them by application count, this query identifies the job with the highest application count.

17. Calculate the percentage of job applications per location.

```
SELECT location, (COUNT(application_id) / (SELECT COUNT(*) FROM applications) * 100) AS application_percentage

FROM jobs j

JOIN applications a ON j.job_id = a.job_id

GROUP BY location;
```

✓ Using joins and a subquery, this query calculates the percentage of applications for each job location, giving insight into application distribution.

18. List job seekers who have applied to multiple jobs in different locations.

```
SELECT seeker_id

FROM applications a

JOIN jobs j ON a.job_id = j.job_id

GROUP BY seeker_id

HAVING COUNT(DISTINCT location) > 1;
```

✓ The query groups applications by `seeker_id` and uses a `HAVING` clause to find job seekers who applied to jobs in more than one location.

19. Find job seekers with the maximum number of applications using a subquery.

```
SELECT seeker_id

FROM applications

GROUP BY seeker_id

HAVING COUNT(application_id) = (

  SELECT MAX(application_count)

  FROM (SELECT COUNT(application_id) AS application_count FROM applications GROUP BY seeker_id) AS seeker_counts

);
```

✓ This query groups applications by `seeker_id` and uses a subquery to identify the job seeker(s) with the highest number of applications, finding those with maximum engagement.

ADVANCED-LEVEL QUESTIONS

20. Find the rank of each job seeker based on the number of applications they submitted, with the highest number of applications ranked first.

```
SELECT seeker_id,
    COUNT(application_id) AS num_applications,
    RANK() OVER (ORDER BY COUNT(application_id) DESC) AS application_rank
FROM applications
GROUP BY seeker_id;
```

✓ This query calculates the total number of applications submitted by each job seeker and assigns a rank based on application count, ranking those with the highest applications first.

21. Retrieve each job's average salary, as well as the difference between each job's salary and the average salary across all jobs.

```
SELECT job_id,
    title,
    job_salary,
    avg_salary_all_jobs,
    (job_salary - avg_salary_all_jobs) AS salary_difference
FROM (
  SELECT job_id,
      title,
      AVG(CAST(SUBSTRING_INDEX(salary_range, '-', -1) AS UNSIGNED)) AS
job_salary,
      AVG(AVG(CAST(SUBSTRING_INDEX(salary_range, '-', -1) AS UNSIGNED))) OVER
() AS avg_salary_all_jobs
  FROM jobs
  GROUP BY job_id, title
) AS job_salary_data;
```

✓ Using a subquery, this query calculates the average salary for each job and compares it with the overall average, showing the salary difference for each job title.

22. Calculate the cumulative number of applications for each job over time, ordered by application date.

```
SELECT job_id,
    application_date,
    COUNT(application_id) OVER (PARTITION BY job_id ORDER BY application_date
ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS
cumulative_applications
FROM applications
ORDER BY job_id, application_date;
```

✓ This query tracks applications over time for each job using a cumulative count with a window function, enabling analysis of how application volume grows over time.

23. Determine the running total of applications submitted by each job seeker, ordered by application date.

```
SELECT seeker_id,
    application_date,
    COUNT(application_id) OVER (PARTITION BY seeker_id ORDER BY
application_date ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS
running_total_applications
FROM applications
ORDER BY seeker_id, application_date;
```

✓ Using a window function, this query calculates a running total of applications submitted by each job seeker, providing insight into each seeker's application activity over time.

24. Rank each job by the number of applications received within each company, with the most-applied-to job ranked first.

```
SELECT j.job_id,
    j.title,
    c.name AS company_name,
    COUNT(a.application_id) AS num_applications,
    DENSE_RANK() OVER (PARTITION BY c.company_id ORDER BY
COUNT(a.application_id) DESC) AS job_rank_within_company
FROM jobs j
JOIN companies c ON j.company_id = c.company_id
JOIN applications a ON j.job_id = a.job_id
GROUP BY j.job_id, j.title, c.company_id, c.name;
```

✓ By partitioning the data by company and applying a dense rank, this query ranks each job within its respective company based on the number of applications, highlighting the most sought-after jobs at each company.