Revilio Take Home

Ayush Sinha

October 2022

Task 1

1.1 Adding Months

Firstly it was required to get the data at the month level. To achieve this I created a temporary table called position_dedup_mnth that contained all the columns of table position_dedup but additionally also contained the columns: Month, Year and Date.
From MySQL perspective to achieve this I used recursive CTE, wherein the temporary table called months recursively calculates and returns a table with all the months within a date range. The date range chosen is such that its lowest value is the lowest among the start dates and highest value is either current date if any of the end dates is null or the highest of the start dates in the position_dedup table.
Once I have the results in the months temporary table I join it with position_dedup table on the condition that the date in the position_dedup table lies within the date found in the months table. This result is stored in the temporary table called position_dedup_mnth.
Note: I could also have done the same operation using cursors and procedure but the query would have been less efficient.

1.1.1 MySQL Code:

```
CREATE TEMPORARY TABLE position_dedup_mnth
with recursive
months (date)
AS
(   #inner query recursively generates dates at month interval for the max range

    SELECT min(STR_TO_DATE(CONCAT(startdate,'-1'),'%Y-%m-%d')) from take_home.position_dedup
    UNION ALL
    SELECT date + INTERVAL 1 Month
    from months
    where date + INTERVAL 1 Month <= (select
                                        case
                                            when (select count(*)
                                        from take_home.position_dedup
                                        where enddate is null) > 0 then CURRENT_DATE()
                                            else
                                        (SELECT max(STR_TO_DATE(CONCAT(enddate,'-1'),'%Y-%m-%d'))
                                            from take_home.position_dedup)
                                        end)
)
#outer query used to find the month name and year of the dates and join with position_dedup table
select     b.*,
           MONTHNAME(date) as Month,
           YEAR(date) as Year,
           date as date
from months as a, take_home.position_dedup as b
where a.date >= STR_TO_DATE(CONCAT(b.startdate,'-1'),'%Y-%m-%d')
and a.date <= (select
                    case
                        when b.enddate is null then current_date()
                        else STR_TO_DATE(CONCAT(b.enddate,'-1'),'%Y-%m-%d')
                    end
               );
```

1.1.2 MySql Output:

```
Here is the illustrative output of the above query:

user_id, position_id, company_id, title, mapped_role, msa, startdate, enddate, Month, Year, date
1,1235,3,Real Estate Salesperson,Salesperson,"New York, NY",2014-06,2020-08,August,2020,2020-08-01
1,1235,3,Real Estate Salesperson,Salesperson,"New York, NY",2014-06,2020-08,July,2020,2020-07-01
1,1235,3,Real Estate Salesperson,Salesperson,"New York, NY",2014-06,2020-08,June,2020,2020-06-01
```

1.2 Identify companies provided by client

I assumed all the companies provided by the client exist in the database. The name of the companies requested by the client were provided as a table called client_requested_companies. This table was natural joined with company_ref table to find the company_id. Once the company_id of the requested companies were identified in the inner query, only those companies were selected from the temporary table position_dedup_mnth created above in the outer query. The results were stored in another temporary table called position_dedup_mnth_name.

1.2.1 MySQL Code:

```
CREATE TEMPORARY TABLE position_dedup_mnth_name
select a.name,b.*
from
    # finding company_id of requested companies
    (select company_id,name
    from take_home.client_requested_companies natural join take_home.company_ref) a,
    position_dedup_mnth b
where a.company_id = b.company_id; #joining with position_dedup_mnth table
```

Note: Skipped providing illustrative output of this query as we are trivially indexing for the companies the client needs.

1.3 Finding the Salaries

Now that we have the employee data of the requested companies at the month level, we have to find their predicted salaries. To do this we join the temporary table created above called position_dedup_mnth_name with the predicted_salaries table on all the parameters the model uses to predict. The result of the query are stored in another temporary table called before_scaling.

1.3.1 MySQL Code:

```
CREATE TEMPORARY TABLE before_scaling
select name, a.msa, Month, a.Year, salary, a.mapped_role
from take_home.predicted_salaries a,position_dedup_mnth_name b
where a.company_id = b.company_id and a.mapped_role = b.mapped_role and a.msa = b.msa and a.year = b.Year;
```

1.3.2 MySql Output:

```
Here is the illustrative output of the above query:

name,       msa,        Month, Year, salary, mapped_role
Netflix,"New York, NY",December,2015,123456,Salesperson
Netflix,"New York, NY",November,2015,123456,Salesperson
```

1.4 Finding the weighted average salary

Now that we have the predicted salaries, I joined the above temporary table with the scaling weights table to calculate the weighted average salary. Weighted average salary was calculated using aggregation functions after grouping by name, msa, month and year. The result were stored in the table Task1_Result.

1.4.1 MySQL Code:

```
CREATE TABLE Task1_Result
select name, msa, Month, Year, sum(salary*weight)/sum(weight) as Average_Salary
from before_scaling natural join take_home.scaling_weights
group by name, msa, Month, Year
order by name, Year;
```

1.4.2 MySql Output:

```
Name,    msa,         Month,    Year,Average_Salary
Netflix,"New York, NY",June,    2014,120000
Netflix,"New York, NY",December,2015,122522.62772844611
```

1.5 Customer Communication

Assumed the client's name to be Nick. The amazon s3 link is denoted as < link >. Here is the email communication:

Hi Nick,

Good Morning. Hope all is well.
Here is the link to a csv file containing the data for the companies that you requested :
< link >
The csv contains 5 columns:

1. Name: The name of the requested company
2. msa: Metropolitan Statistical Area of the company
3. Month: The month of the reported average salary for the company
4. Year: The year of the reported avrage salary for the company
5. average_salary: The average salary calculated for the company

This csv is ordered by Name of the company and the year. Let me know if this fulfills the requirement or if there is anything amiss. Looking forward to hearing from you!

Regards,
Ayush Sinha
Revilio Labs

Task 2

2.1 Bringing Education_dedup to month level

For the task of identifying contingent workers I assumed that whoever is working as well as simultaneously studying should be an intern or part-time worker. This assumption has its shortcomings such as:

1. Employee who are working without simultaneously pursuing education can also work part-time.
2. Employee's can work full time and pursue education part-time

Now the task was to find the intersection between an employee working and whether he was simultaneously studying at a month level. To do this the education_dedup table had to be brought to the month level first. This was done using similar logic as outlined in 1.1. The results were stored in a a temporary table called education_dedup_mnth.

2.1.1 MySQL Code:

```
CREATE TEMPORARY TABLE education_dedup_mnth
with recursive
months (date)
AS
(
    #inner query recursively generates dates at month interval for the max range
    SELECT min(STR_TO_DATE(CONCAT(startdate,'-1'),'%Y-%m-%d')) from take_home.education_dedup
    UNION ALL
    SELECT date + INTERVAL 1 Month
    from months
    where date + INTERVAL 1 Month <= (select
                                        case
                                            when (select count(*)
                                        from take_home.education_dedup
                                        where enddate is null) > 0 then CURRENT_DATE()
                                            else
                                        (SELECT max(STR_TO_DATE(CONCAT(enddate,'-1'),'%Y-%m-%d'))
                                            from take_home.education_dedup)
                                        end)
)
#outer query used to find the month name and year of the dates and join with education_dedup
select     b.*,
           MONTHNAME(date) as Month,
           YEAR(date) as Year,
           date as date
from months as a, take_home.education_dedup as b
where a.date >= STR_TO_DATE(CONCAT(b.startdate,'-1'),'%Y-%m-%d')
and a.date <= (select
                    case
                        when b.enddate is null then current_date()
                        else STR_TO_DATE(CONCAT(b.enddate,'-1'),'%Y-%m-%d')
                    end
               );
```

2.1.2 MySql Output:

```
user_id, school, degree, startdate, enddate, Month, Year, date
1,Sunnyside High School,High School,2010-09,2014-06,September,2010,2010-09-01
1,Sunnyside High School,High School,2010-09,2014-06,October,2010,2010-10-01
1,Sunnyside High School,High School,2010-09,2014-06,November,2010,2010-11-01
```

2.2 Finding Employees working part time

To find the intersection between employees who were simultaneously holding positions in companies as well as pursuing an education, I utilized the temporary table: position_dedup_mnth_name created in task 1 and left joined it with education_dedup_mnth on year, month and user_id. All the columns of position_dedup_mnth_name were kept but only user_id of education_dedup_mnt was taken(renamed as intern)
If a employee working in a certain position wasn't pursuing an education simultaneously his intern field would come out as null. This fact was utilized in the outer query to create a new temporary table called part_time which had a column named employee_type whose value was 1 if the employee was holding a position as well as pursuing an education and 0 if he wasn't pursuing a education simultaneously.

2.2.1 MySQL Code:

```
create temporary table part_time
with emp_type
as
(
    select a.* ,b.user_id as intern
    from position_dedup_mnth_name a left join education_dedup_mnth b on
    (a.Year = b.Year and a.Month = b.Month and a.user_id = b.user_id)
)
select NAME, COMPANY_ID, MAPPED_ROLE, MSA, MONTH, YEAR,IF(intern is not null, 1, 0) as employee_type
```

```
from emp_type;
```

2.2.2 MySQL Output:

```
Name, Company_id, mapped_role, msa, Month, Year, employee_type
Netflix,3,Salesperson,"New York, NY",August,2014,0
Netflix,3,Salesperson,"New York, NY",September,2014,1
```

2.3 Find the predicted salaries

As the model predicting salaries doesn't take employee_type as an input while predicting salaries I assumed that the employee_type was known to the model before hand and that the original combination of columns would suffice.
A similar query to 1.3.2 was written additionally adding column employee_type. The results were stored in temporary table called before_scaling_2.

2.3.1 MySQL Code:

```
CREATE TEMPORARY TABLE before_scaling_2
select name, a.msa, Month, a.Year, salary, a.mapped_role, b.employee_type as Employee_type
from take_home.predicted_salaries a,part_time b
where a.company_id = b.company_id and a.mapped_role = b.mapped_role and a.msa = b.msa and a.year = b.Year;
```

2.3.2 MySQL Output:

```
Name,   msa,          month,   year, salary, mapped_role, employee_type
Netflix,"New York, NY",September,2015,120000,Software Engineer,0
Netflix,"New York, NY",September,2015,123456,Salesperson,1
```

2.4 Finding the weighted average salary

Once we had the predicted salaries of the employee along with the employee_type column at the month level. To calculate the average weighted salary, steps similar to section 1.4 were taken. The difference was the we were additionally grouping by Employee_Type. Employee_type value 0 was renamed as 'full-time' and 1 was renamed as 'part-time'.
The results were stored in Table called Task2_Result

2.4.1 MySQL Code:

```
CREATE TABLE Task2_Result
select name, msa, Month, Year,  IF(Employee_Type=1,'part-time','full-time') as Employee_Type,
sum(salary*weight)/sum(weight) as average_salary
from before_scaling_2 natural join take_home.scaling_weights
group by name, msa, Month, Year, Employee_Type
order by name,Year;
```

2.4.2 MySQL Output:

```
Name, msa, Month, Year, Employee_Type, average_salary,
Netflix,"New York, NY",April,2015, full-time, 120000
Netflix,"New York, NY",April,2015, part-time, 123456
```

2.5 Customer Communication

Here is the email communication with Nick. The new csv file is similarly stored in Amazon s3 with <link> denoting the URL link to it.

Hi Nick,
Good Morning. Glad to know that the data provided was useful! I was able to split the average salary calculation using the two types of workers. I did this using the fact that part-time/interns are primarily those employee's of the company who concurrently pursue education.
Here is the link to a csv file containing the data in the new format:
<link>
The csv contains all the columns previously outlined and additionally has a new column called Employee_Type which identifies whether the average salary reported for the company is for part-time or full-time employees. Let me know if this fulfills your requirement!

Regards,
Ayush Sinha
Revilio Labs