Branch

I. Introduction

Branch is the premier website to explore job postings in the context of company and employee information across over 100 industries. While a handful of career websites contain subsets of this information, none combine them all the way Branch does. LinkedIn and Handshake contain job postings and company information; TheOrg contains company and employee information; and Glassdoor contains salaries. Branch is a one-stop-shop for career exploration and networking by combining these functions.

Branch contains four main pages. On our Find Jobs page, users may search for open positions by position name and view the job description, our estimated salary, and similar jobs to explore. On our Companies page, users can look up companies to find a company description as well as the company leadership. The Find Companies page serves as an additional way to search for companies, this showing companies with specific job openings. Finally, on the Find Employees page, users can search for people by name, then view a description of the employee in addition to a list of similar people as networking suggestions.

Branch is developed by Kat Wang (katawang@seas.upenn.edu, kattawang), Angelina Heyler (aheyler@seas.upenn.edu, aheyler), Angie Shen (angelias@seas.upenn.edu, shen-angie), and Spencer Solit (ssolit@seas.upenn.edu, ssolit).

II. Architecture

Our product leverages several industry standard technologies. Our database is hosted on Amazon AWS using a RDS instance so that we could easily set up, operate, and scale our data management. We query data through a Node.js backend using SQL queries, allowing our service to collect relevant data on demand for users. Our web app is constructed using a React framework to render data in an understandable form for users and allow easy interaction to find data.

III. Data

Our app queries data from three datasets: (1) web-scraped job posting and company data from Handshake, (2) web-scraped organizational charts from TheOrg, and (3) a Kaggle salaries dataset.

(1) Web-scraped Handshake data

Description: This dataset contains job postings from Handshake. These were scraped manually using fetch.js requested and cleaned before entry into the database.

Summary statistics and EDA: There are 37,803 job postings in our scraped dataset with 6,190 employers represented. Each employer has an average of 6 postings, with a median of 2 postings, lower quartile of 2 postings, upper quartile of 5 postings, and a maximum of 318. Of these postings, 22,862 are job postings

and 14,941 are internship postings. From the 5 locations listed, there are 36k jobs with at least one location listed, 7k have at least two cities listed, 5k have at least three cities listed, 3.6k have at least four cities listed, and 3k have at least five cities listed.
Explanation: We used this data to display job postings for companies so that users can browse open positions. This is an important component of our platform's goal of career exploration — to be able to link people and companies with open positions for users to apply.

(2) Web-scraped TheOrg data
Description: This dataset contains ~300k companies and org charts that show the managerial levels of the employees working at a company. These were scraped manually using fetch.js requests.
Summary stats: There are 41,941 unique companies in the scraped TheOrg data. Around half are small companies, a quarter are medium-sized, and a quarter are large: 20,512 come from employee size ranges of 10-50, 11,871 have 50-200 employees, and 9,558 have more than 500 employees. The top states represented include California, New York, and Texas, with a table in Appendix 1.

There are 689,702 unique employees represented in TheOrg dataset. The employees listed tend to be higher ranking members of the company, as the top 10 roles include Director, Board Member, CEO, COO, and CTO. This could also be because these roles exist for all companies, whereas positions such as "software engineer" would mainly occur at tech companies. The organizational chart data (i.e., links from individuals to their bosses) includes 442,028 unique employees as subordinates, or ~64% of all employees in the dataset.

Explanation: We used this data to display information about current company leadership and employers, show connections between employees, and recommend people in the same industry as the user.

(3) Kaggle STEM salaries data (link)
Description: The third dataset we used is called "Data Science and STEM Salaries." This is a Kaggle dataset of salary information that was scraped from levels.fyi in 2017. The dataset contains information such as job title, job location, years of experience, education level, and compensation (base salary, bonus, stock grants).
Summary Statistics: The dataset contains 62,000 salary records from top companies. Upon conducting exploratory data analysis, we found that the top employers in this dataset include Amazon, Microsoft, Google, and Facebook, with a table of top 10 employers in Appendix 2.1. We also explored the distribution of total yearly compensation, base salary, stock grant value, and years of experience in Appendix 2.2, as well as categorical variables denoting highest education received and race in Appendix 2.3.

Explanation: We used this data for job salary information based on position and level, as well as to predict the salaries of job postings.

IV. Database

a) Data Ingestion & Entity Resolution Efforts

Handshake Data Ingestion. Our scraped Handshake data consisted of one large JSON file with a series of job postings. As a result, the first step in ingesting the data was to unnest the file and pull out relevant information, including job information (e.g., id, posting date, employer name/id, expiry date), then nested job information (e.g., duration, industry name/id, text description, title) and location information (e.g., city, state, country). Next, from manual inspection, we found that there was a split between using “United States” vs. “United States of America” in country names, so we fixed the names to all be consistent. Because each job posting could also include multiple locations, we also unnested these and kept the cities, states, and countries with the first five locations (there were only a handful of postings with more than 5 locations). Finally, we completed smaller touches such as casting dates to pandas.datetime values

TheOrg Data Ingestion. Our scraped data from TheOrg included a series of JSONs, each including a chunk of OrgCharts from companies from a particular country. First, we created a companies CSV that listed the company name, social media links, location, employee size range, and description. This was done by iterating through all of the OrgCharts and only pulling out the company information (i.e., not touching employee information yet). Here, we found that many states were missing in the states column but appeared in the full location text, so we wrote a function to fill state abbreviations in our state column. Other entries had information in the individual city, state, country columns but not in the locationString column, so we also filled those in. For files that came from international companies, we found different issues with the location columns. For example, for Chinese locations, there were duplicate names for many cities/provinces, such as “Shanghai” and “Shanghai Shi,” since “shi” means city in Chinese. The same issue occurred with provinces. To resolve this, we stripped these modifiers on the city/province names. We also removed leading/trailing space characters that might prevent matches. Finally, we concatenate the results across all the input files, drop duplicate companies, and save the CSV.

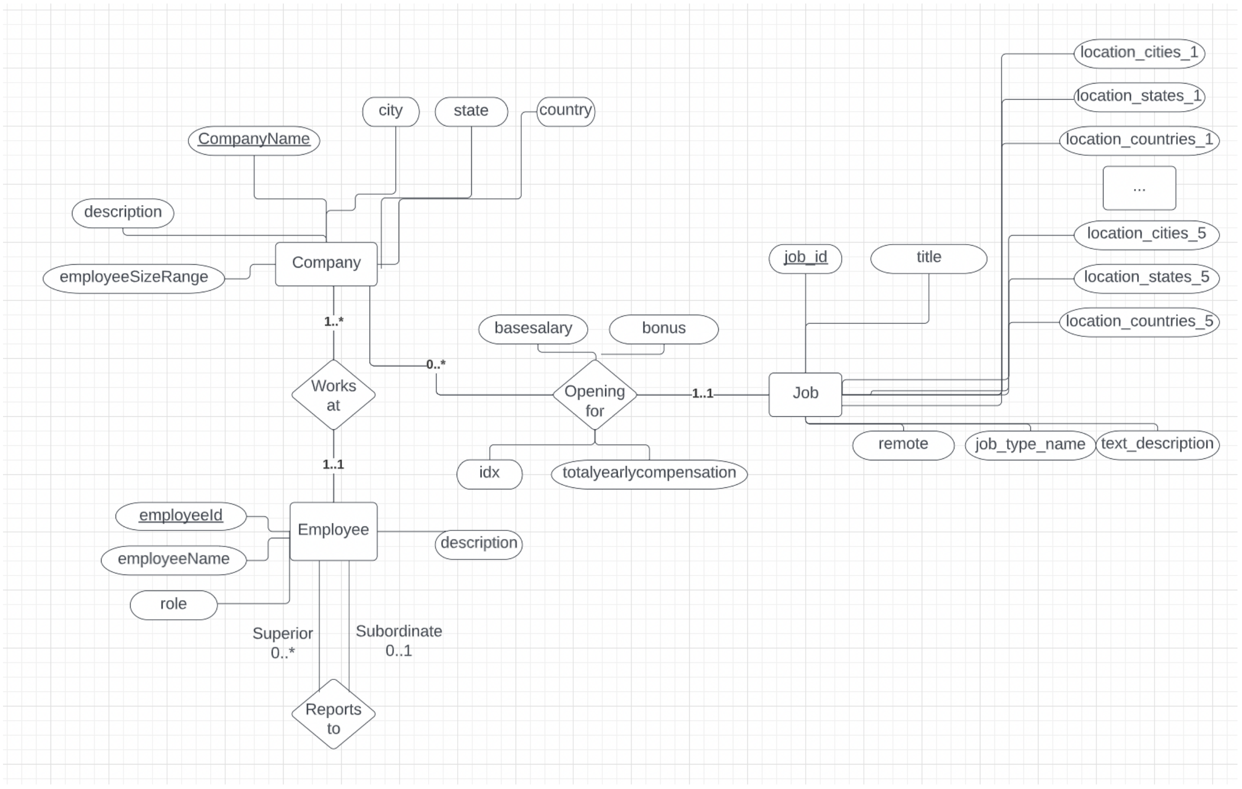
Next, we created our OrgChart CSV. Here, we unnested the series of JSONs by diving into the employeeNodes and pulling out relevant employee information. First, we noticed that employee IDs sometimes contained “p-” before their numbers. This was inconsistent with parentId, which references id and only consisted of numbers. As a result, we cleaned the id column to match. Finally, we saved just the employeeId and parentId columns.

Finally, we created our employee CSV. To do so, we unnested employee information such as their name, role, role function, description, whether they’re remote and also included the company name as a foreign key referencing companies. This information was relatively clean.

Kaggle Salary Data Ingestion. To clean the Kaggle dataset, we first fixed discrepancies in company names (e.g., changing “MSFT” to “Microsoft”) and by capitalizing the names of companies for consistency. Given location strings, we also parse these into city, state, and country values for consistency with our other schemas. Finally, we drop unnecessary columns and save the results.

Entity Resolution Efforts. Between 6,190 companies on Handshake and 42,031 from TheOrg, we were able to match 416 companies on exact matches. We also used fuzzy string matching via the Python library fuzzywuzzy to score potential matches, which took a runtime of 14 hours, then experimented with a series of different thresholds to optimize matches. However, we found that even with high score thresholds, the matches were not true matches — they included acronyms with one or more differing letters and companies with structurally similar but semantically different words (e.g. “Banking” and “Baking”). We manually went through a list of around 300 potential matches as found by the fuzzy matching and found an additional 24 matches. In total, we could match 440 companies, though upon analyzing these matches, they were generally larger, more well-known companies, which was most relevant for job postings and salary information.

b) ER Diagram:



c) Number of Instances per Table:

Table Name	# Instances
HS_Jobs	37,803
TO_Employees	689,702
TO_Companies	42,095
TO_OrgChart	442,028
Salary	62,642

d) Normal Form and Justification: Functional dependencies for each relation:

$F_{TO\_companies}$	$F_{TO\_Employees}$	F_{HS_Jobs}	F_{Salary}
CompanyName → description	employeeId → employeeName	job_id → job_name	company, title, idx → totalyearlycompensation
CompanyName → employeeSizeRange	employeeId → role	job_id → location_cities_1	company, title, idx → bonus
CompanyName → city	employeeId → parentId	job_id → location_states_1	company, title, idx → basesalary
CompanyName → state		job_id → location_countries_1	
CompanyName → country		...	
		job_id → location_cities_5	
		job_id → location_states_5	
		job_id → location_countries_5	
		job_id → remote	
		job_id → job_type_name	
		job_id → text_description	

Proving BCNF:
Candidate key for TO_companies = CompanyName. Note that every functional dependency here depends on CompanyName alone, so this is in BCNF.
Candidate key for TO_employees = employeeId. Once again, every functional dependency depends on employeeId, so this is in BCNF.
Candidate key for HS_jobs = job_id. This determines everything, so this is in BCNF.
Candidate key for Salary = (company, title, idx). All of the functional dependencies have X= this key, so this is also in BCNF.

V. Queries

Complex Queries:

1. *Find better-paying jobs for a person's function.* Compensation can be an important factor to users when considering whether to stay in their current role or apply to new roles. A person might like to know if there are better-paying job openings out there for their same role. This can give them insight to help negotiate a raise at their current role, or give them a new opportunity to apply for upwards mobility. Given a person, this query finds the highest paying open position with the same title as that person. We use this as part of the job page.

```
WITH eq_role_postings AS
    (SELECT open_sals.role, job_id, salary, company
        FROM (SELECT *
            FROM (SELECT role
                  FROM TO_Employees
                  WHERE employee_id = ${employee_id}) employee_reduce
                  Natural JOIN
                  (SELECT Salary.title AS role, company,
                      MAX(basesalary + bonus) AS salary
                   FROM Salary
                   GROUP BY title, company) sal_reduce) open_sals
                   JOIN
                   (SELECT HS_Jobs.title AS role, HS_Jobs.id AS job_id,
employer_name
                    FROM HS_Jobs) job_reduce
                   ON company=employer_name AND open_sals.role=job_reduce.role)
SELECT *
FROM eq_role_postings
WHERE salary >= ALL (
    SELECT salary
    FROM eq_role_postings
);
```

2. *Company leadership.* Select the people who are part of a company's leadership—that is, people who are at most 3 degrees of freedom away from the CEO. Because leadership is an important part of understanding a company's culture and governance, users might like to be able to view the top ranking people in a company from the company page. The SQL queries is as follows:

```
WITH CompanyCEO AS (
   SELECT employee_id
   FROM TO_Employees
   JOIN TO_companies ON TO_Employees.CompanyName = TO_companies.CompanyName
   WHERE TO_companies.company_id = ${inputSearch} AND TO_Employees.role LIKE
'%CEO%'
   ),
   Dof1 AS (
       SELECT worksUnder.employee_id
       FROM worksUnder
       JOIN CompanyCEO ON worksUnder.parent_id = CompanyCEO.employee_id
   ),
   Dof2 AS (
       SELECT worksUnder.employee_id
       FROM worksUnder
       JOIN Dof1 ON Dof1.employee_id = worksUnder.parent_id
   ),
   Dof3 AS (
       SELECT worksUnder.employee_id
       FROM worksUnder
       JOIN Dof2 ON Dof2.employee_id = worksUnder.parent_id
       ),
   AllDof3Employees AS (
       SELECT * FROM CompanyCEO
       UNION ALL (SELECT * FROM Dof1)
       UNION ALL (SELECT * FROM Dof2)
       UNION ALL (SELECT * FROM Dof3)
   )
SELECT DISTINCT employeeName AS Name, role AS Role, description AS Description
FROM TO_Employees
JOIN AllDof3Employees ON TO_Employees.employee_id =
AllDof3Employees.employee_id;
```

3. *Similar people recommendation for networking*. To recommend similar people that might be helpful for networking, we also wrote a query to return 5 people within 2 degrees of freedom from the person of interest. This is implemented as part of the employees page, where if you click on a specific employee, a table of similar people also appears on the person's profile page, along with a description of them. The SQL query is as follows:

```
WITH desiredRole AS (
                  SELECT role
                  FROM TO_Employees
                  WHERE employee_id = ${inputPerson}
                  ),
                  deg0 AS (
                  SELECT employee_id
                  FROM TO_Employees JOIN desiredRole d ON TO_Employees.role =
d.role
                  LIMIT 5
                  ),
                  deg1 AS (
                      SELECT employee_id
                      FROM (SELECT deg0.employee_id FROM worksUnder JOIN deg0
ON deg0.employee_id = worksUnder.parent_id) boss
                      UNION ALL (SELECT deg0.employee_id FROM worksUnder JOIN
deg0 ON deg0.employee_id = worksUnder.employee_id)
                      LIMIT 5
                  ),
                  deg2 AS (
                      SELECT employee_id
                      FROM (SELECT deg1.employee_id FROM worksUnder JOIN deg1
ON deg1.employee_id = worksUnder.parent_id) boss
                      UNION ALL (SELECT deg1.employee_id FROM worksUnder JOIN
deg1 ON deg1.employee_id = worksUnder.employee_id)
                      LIMIT 5
                  ),
                  alldegs AS (
                          SELECT * FROM deg0
                          UNION ALL (SELECT * FROM deg1)
                          UNION ALL (SELECT * FROM deg2)
                  )
              SELECT DISTINCT (TO_Employees.employee_id) AS employee_id,
employeeName, CompanyName, role
              FROM TO_Employees
              JOIN alldegs ON TO_Employees.employee_id = alldegs.employee_id
              ORDER BY employeeName
              LIMIT 5
```

4. *Job search for fully in-person companies.* One important factor for job searches post-pandemic is understanding whether companies are operating fully in-person, or if some people are still remote. In this case, we might like to know what jobs are available at companies that are fully back in-person—that is, find jobs at companies with no remote employees. We use this as a filter on our job search page. The query is below:

```
SELECT id, employer_name, title,
                CASE
                  WHEN remote=0 THEN "No"
                  WHEN remote=1 THEN "Yes"
                  ELSE "Unknown"
                END AS remote
FROM HS_Jobs H
WHERE H.employer_name NOT IN (
      SELECT CompanyName
      FROM TO_Employees
      WHERE remote = 1
) AND H.remote = 0 AND H.title LIKE "%${inputSearch}%"
ORDER BY employer_name;
```

Simple Query:

5. *Search for jobs at a location where the employer has high salaries.* When looking for potential companies you'd like to work for, you might want to be able to search for a list of companies and position titles where the average salary exceeds your threshold. In this query, we select job and job titles from open positions on Handshake, where the user searches for jobs in a specific location

and wants the average salary at that company to exceed some threshold. The query is used on the Find Companies With Jobs page, where users search for companies with jobs meeting the salary and location criteria:

```
WITH ExpSalaryTable AS (SELECT company, AVG(totalyearlycompensation) AS
AvgSalary
                        FROM Salary
                        GROUP BY company
                        HAVING AvgSalary > 10000)
SELECT employer_name AS Company, title AS JobName
FROM HS_Jobs
JOIN ExpSalaryTable ON HS_Jobs.employer_name = ExpSalaryTable.company
WHERE (location_cities_1 LIKE 'New York'
    OR location_cities_2 LIKE 'New York'
    OR location_cities_3 LIKE 'New York'
    OR location_cities_4 LIKE 'New York'
    OR location_cities_5 LIKE 'New York');
```

VI. Performance evaluation

Event	**Before Optimization** (Queries in Appendix 3)	**After Optimization**	**Input**
Query #1 (Complex)	30s	1s	4640
Query #2 (Complex)	19s	6s	743
Query #3 (Complex)	13s	1s	275
Query #4 (Complex)	11s	3s	“Business Analyst”
Query #5 (Simple)	908 ms	429 ms	“New York,” 10,000

Query #1 Optimization: This query takes in a value of employee_id in the call, so we experimented by randomly selecting one employee_id then timing the query across different implementations. We optimized this query by pushing selections and projections down and adding a group-by to remove duplicate (title, company) values.
Query #2 Optimization: To optimize this query, we added an index to the worksUnder relation of (parent_id, employee_id). This helps the database quickly look up values by parent and employee id, which is done repeatedly in this query when looking for employees working under a particular employee. We also changed the three union operations to “union all,” which prevents the database from having to eliminate duplicates three separate times, and instead include a “distinct” in the main select statement. For timing purposes, we used the example company “Neuralink.”
Query #3 Optimization: Since this query takes in a specific employee_id as an input, we tested performance by randomly selecting an employee_id (275) and timed different versions of the query with this same value. We optimized this query by changing unions

to union ALL, which means that the database doesn’t need to sort through the common table expressions to remove duplicates at each step.
Query #4 Optimization: We optimized this query by removing a CTE for InPersonCompanies that grouped by company, summed up the number of remote employees and checked that the sum was 0, then joining our handshake data with this table. This was slow because it involved an aggregation operation in the CTE, then an additional join on the employer names, whereas the uncorrelated query involves just a selection and checking for existence over a minimally sized (distinct) relation of company names. This was timed using New York as the location.
Query #5 Optimization: Although this was a simple query, we optimized it moderately by using inner joins instead of “where HS_Jobs.employer_name = ExpSalaryTable.company” and by pushing down projections to the CTE level. This was timed with “New York” as the city and 10,000 as the threshold.

VII. Technical challenges

We encountered several technical challenges over the course of this project. The first came with data collection through web scraping. Initially concerned with results being too large and overloading the website, we initially focused on scraping smaller companies from TheOrg. However, when it came to matching entities across TheOrg, Handshake, and our salary dataset, we found that TheOrg matched with only 44 company names, since these smaller companies tended to be less well-known and also have fewer positions open (if any) on Handshake. As a result, we went through a second round of web scraping, to expand our dataset, but TheOrg had somewhat changed their website structure, so we had to make edits to the scraping code.

Next, we had to undergo extensive data cleaning for the datasets to be usable with connected entities among them. As described in detail in Section IV, we needed to remove or fix incorrect/null values, attempted fuzzy matching and manual inspection for entity matching due to slightly different company names across different sites, and aggregated datasets from manually scraped chunks of datasets, which had slightly different cleaning issues for different company countries.

Finally, because we were using an older version of Node (0.1.0), which required a legacy provider to start, we encountered lots of issues with getting Node to start and downloading appropriate versions of packages, such as ant design, express, and mocha.

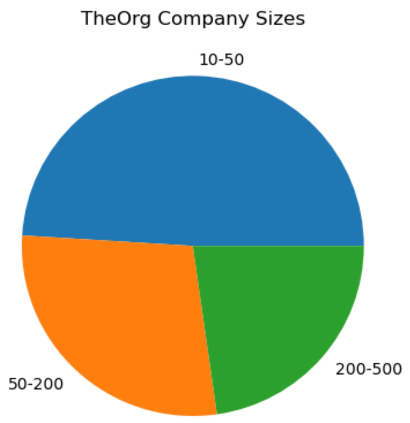
VIII. Credits

We used the code from Exercise3 and HW2 as a starting point for building our application. We used a branch icon from Icons8 for our website icon.

Appendix

Appendix 1: TheOrg data descriptive statistics: top 10 states represented in the companies data (left), size ranges of companies as measured by number of employees (right).

State	Number of Companies
California	9,501
New York	4,419
Texas	2,486
Massachusetts	2,224
Florida	1,701
Illinois	1,425
Washington	1,243
Virginia	1,204
Pennsylvania	1,176
Colorado	965

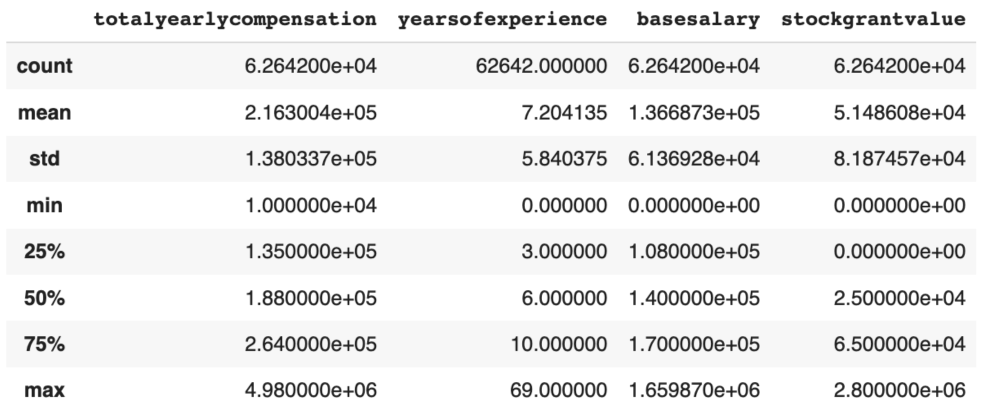


Appendix 2: Kaggle salary data descriptive statistics.
2.1 Top 10 employers in the salary dataset.

	Employer	# Employees in Dataset
1	Amazon	8,126
2	Microsoft	5,216
3	Google	4,330
4	Facebook	2,990
5	Apple	2,028
6	Oracle	1,128
7	Salesforce	1,056

8	Intel	949
9	Cisco	907
10	IBM	907

2.2 Distribution of total yearly compensation, years of experience, base salary, and stock grant value.



	totalyearlycompensation	yearsofexperience	basesalary	stockgrantvalue
count	6.264200e+04	62642.000000	6.264200e+04	6.264200e+04
mean	2.163004e+05	7.204135	1.366873e+05	5.148608e+04
std	1.380337e+05	5.840375	6.136928e+04	8.187457e+04
min	1.000000e+04	0.000000	0.000000e+00	0.000000e+00
25%	1.350000e+05	3.000000	1.080000e+05	0.000000e+00
50%	1.880000e+05	6.000000	1.400000e+05	2.500000e+04
75%	2.640000e+05	10.000000	1.700000e+05	6.500000e+04
max	4.980000e+06	69.000000	1.659870e+06	2.800000e+06

2.3 Distribution of binary variables for highest education level received and race.

Attribute Name	**Number of 0's**	**Number of 1's**
Highschool	62322	320
Bachelors_Degree	50037	12605
Masters_Degree	47251	15391
Doctorate_Degree	60839	1803
Race_Asian	50870	11772
Race_White	54610	8032
Race_Two_Or_More	61838	804
Race_Black	61952	690
Race_Hispanic	61512	1130

Appendix 3: Unoptimized Queries

Query #1:

```
WITH eq_role_postings AS (
    SELECT job_id, TO_Employees.role AS role, salary, company, employee_id
    FROM ((SELECT HS_Jobs.id AS job_id, HS_Jobs.title AS role, basesalary +
bonus AS salary, company
          FROM HS_Jobs JOIN Salary S
            ON HS_Jobs.title = S.title) open_sals
        JOIN TO_Employees ON TO_Employees.role = open_sals.role)
)
SELECT job_id
FROM eq_role_postings
WHERE salary >= ALL (
    SELECT salary
    FROM eq_role_postings);
```

Query #2:

```
WITH CompanyCEO AS (
   SELECT employee_id
   FROM TO_Employees
   WHERE CompanyName LIKE ${inputSearch} AND role LIKE '%CEO%'
   ),
   Dof1 AS (
       SELECT worksUnder.employee_id
       FROM worksUnder
       JOIN CompanyCEO ON worksUnder.parent_id = CompanyCEO.employee_id
   ),
   Dof2 AS (
       SELECT worksUnder.employee_id
       FROM worksUnder
       JOIN Dof1 ON Dof1.employee_id = worksUnder.parent_id
   ),
   Dof3 AS (
       SELECT worksUnder.employee_id
       FROM worksUnder
       JOIN Dof2 ON Dof2.employee_id = worksUnder.parent_id
       ),
   AllDof3Employees AS (
       SELECT * FROM CompanyCEO
       UNION (SELECT * FROM Dof1)
       UNION (SELECT * FROM Dof2)
       UNION (SELECT * FROM Dof3)
   )
SELECT employeeName AS Name, role AS Role, description AS Description
FROM TO_Employees
JOIN AllDof3Employees ON TO_Employees.employee_id =
AllDof3Employees.employee_id;
```

Query #3:

```
WITH deg0 AS (
    SELECT employee_id
    FROM TO_Employees
    WHERE role = @search_position
    LIMIT 5
    ),
    deg1 AS (
        SELECT deg0.employee_id
             FROM worksUnder
             JOIN deg0
             ON deg0.employee_id = worksUnder.parent_id) boss
        UNION (SELECT deg0.employee_id
               FROM worksUnder
               JOIN deg0
               ON deg0.employee_id = worksUnder.employee_id)
        LIMIT 5
    ),
    deg2 AS (
        SELECT deg1.employee_id
         FROM worksUnder
         JOIN deg1
         ON deg1.employee_id = worksUnder.parent_id) boss
        UNION
             (SELECT deg1.employee_id
             FROM worksUnder
             JOIN deg1
             ON deg1.employee_id = worksUnder.employee_id)
        LIMIT 5
    ),
    alldegs AS (
            SELECT * FROM deg0
            UNION (SELECT * FROM deg1)
            UNION (SELECT * FROM deg2)
    )
SELECT employeeName
FROM TO_Employees
JOIN alldegs ON TO_Employees.employee_id = alldegs.employee_id
ORDER BY employeeName
LIMIT 5;
```

Query #4:

```
WITH InPersonCompanies AS (
   SELECT *
   FROM TO_Employees
   GROUP BY CompanyName
   HAVING SUM(remote) = 0
)
SELECT *
FROM HS_Jobs H
JOIN InPersonCompanies
```

```
ON H.employer_name = InPersonCompanies.CompanyName
WHERE H.remote = 0
AND H.title LIKE @title
ORDER BY employer_name;
```

Query #5:

```
WITH ExpSalaryTable AS (SELECT company, AVG(totalyearlycompensation) AS
AvgSalary
                       FROM Salary
                       GROUP BY company
                       HAVING AvgSalary > 10000)
SELECT DISTINCT employer_name AS Company, title AS JobName
FROM HS_Jobs, ExpSalaryTable
WHERE HS_Jobs.employer_name = ExpSalaryTable.company
AND (location_cities_1 LIKE 'New York'
   OR location_cities_2 LIKE 'New York'
   OR location_cities_3 LIKE 'New York'
   OR location_cities_4 LIKE 'New York'
   OR location_cities_5 LIKE 'New York');
```