INFO-6210 DATA MANAGEMENT AND DATABASE DESIGN

**JOBS DATABASE**

**ABSTRACT:**

Looking for jobs and internships seems a task of its own and the search is no longer on sole fulfilment of the required skills, but a lot of networking and recommendations is involved around too. The amount of work involved in finding the correct job builds a great amount of anxiety among the job seekers and the recruiters who want the right talent for their company. First, matching the job seekers with the right employers and second, provide guidance to aspiring job seekers on the skills that are in demand so that they can build them to stay relevant in the job market. The job providers and job seekers form a large amount of data which provides for many interesting trends for analysis and interpretation to make the most of data available. The presence of information on job skills, salaries and user tendencies in many existing websites such as Glassdoor, LinkedIn and many other websites can be utilized to match people to the suitable positions. The jobs database would be a one stop solution to reduce the job search and talent acquisition stress levels.

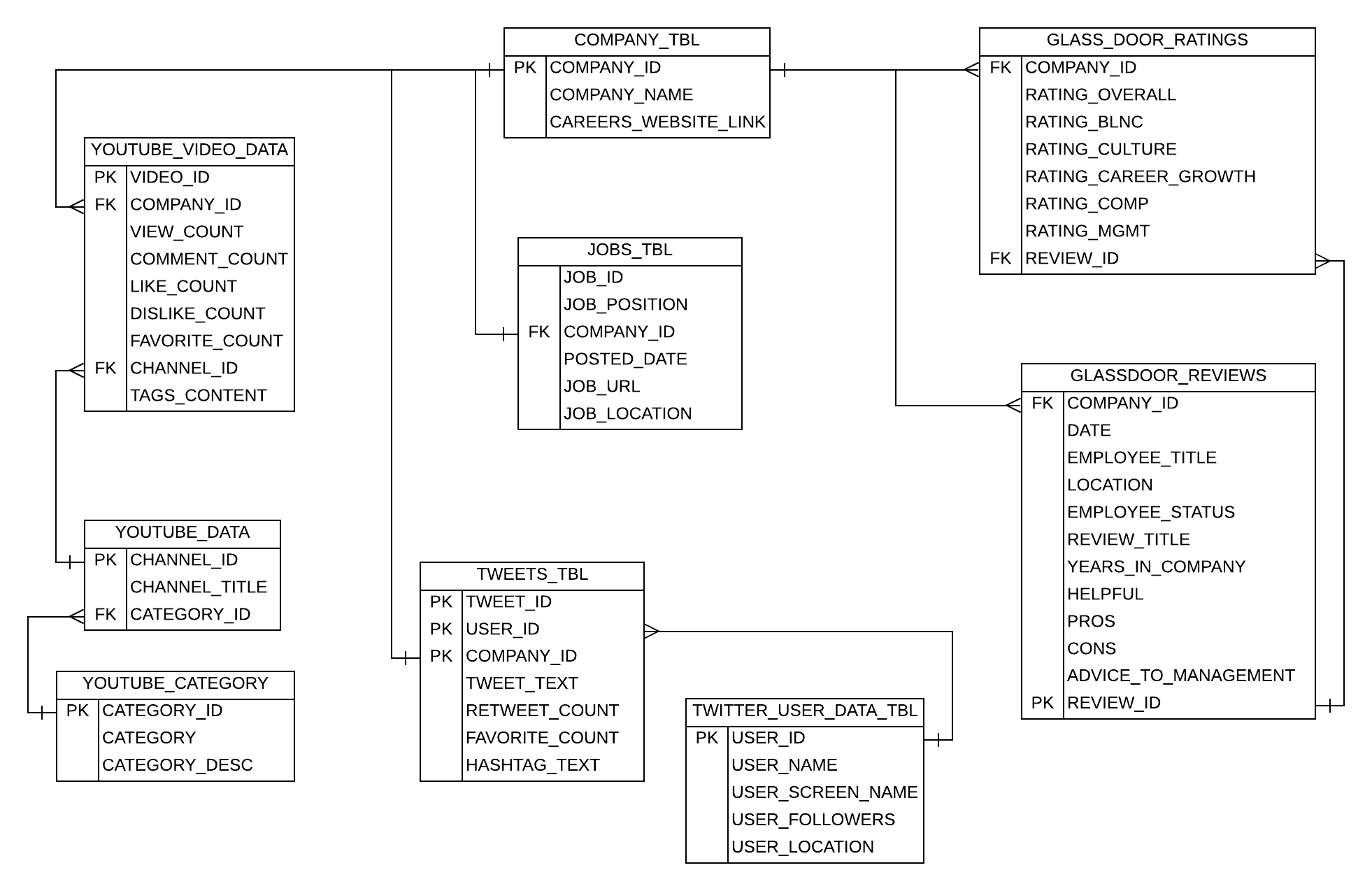
**INTRODUCTION**:

Job searching is the most important phase in anyone’s life. A right job can make someone’s life to the best than he could ever had imagined. In this present internet era, there are many websites that are posting the numerous jobs and the specific skill sets required for those jobs.

**DATA:**

The first step in any database project is the getting the necessary data. The data which we want can be collected from various sources. Now there are many online platforms which can provide you the necessary data for the creation of the project. First, we collected the list of 300 companies in finance domain and then we collected the data from those company websites using the web scraping. We also scraped the Glassdoor website and collected company reviews and ratings for about 28 companies from the list. We also scraped Twitter data and YouTube data using their API’s collected the relevant tweets and video data.

# **ER DIAGRAM:**

****

DESCRIPTION ABOUT TABLES:

There are about nine tables in the JOBSDB\_DMDD database.

* **COMPANY\_TBL**
* **JOBS\_TBL**
* **TWEETS\_TBL**
* **TWITTER\_USER\_DATA\_TBL**
* **YOUTUBE\_DATA**
* **YOUTUBE\_CATEGORY**
* **YOUTUBE\_VIDEO\_DATA**
* **GLASSDOOR\_RATINGS**
* **GLASSDOOR\_REVIEWS**

**1) COMPANY\_TBL (company table):**

This table contains the following attributes:

**COMPANY\_ID(PK):** As we collected the data of the 300 finance company’s we created a unique id for each of the company. This column is the primary key of the table and each row can be uniquely identified using this primary key.

**COMPANY\_NAME:** This attribute or the column contains all the company names which we collected for the finance department.

**CAREERS\_WEBSITE\_LINK:** Each company has its own website and it has its careers page through which one can apply for the job. We collected the career page links for each of the 300 companies.

**JOBS\_TBL:**

This table has the following attributes:

**JOB\_ID:**

When we check for any company website for the jobs we can see that there will be a job id uniquely defined for each and every different type of the job they post. This JOB\_ID refers to the same thing.

**JOB\_POSITION:**

Job position refers to the what type of position he is applying for. For example, he may apply to software Engineering, Data scientist etc.

**COMPANY\_ID:**

This is the foreign key in this table which refers to the primary key in the COMPANY\_TBL.

Foreign keys are used to provide the perfect link between the tables.

**POSTED\_DATE:**

Posted date refers to on which date the job position is posted on the website.

**JOB\_URL:** Job URL refers to the link to apply for that position. We go to any careers website we will see that click here to apply and after clicking that we will be redirected to the new page. This job\_url stores the link of the page.

**JOB\_LOCATION**: Job location refers to on which location this job is available.



TWEETS\_TBL:

This table has the following attributes:

The primary key in this table is the composite key i.e which is the combination of TWEET\_ID, USER\_ID, COMPANY\_ID. The combination of this three can identify each row of uniquely without any ambiguity.

**TWEET\_ID:** When the user posts any tweet in twitter it will generate a unique id for each and every tweet. This unique id is the TWEET\_ID.

**USER\_ID:** Each person who has a twitter account will be given a unique id. This id is the USER\_ID.

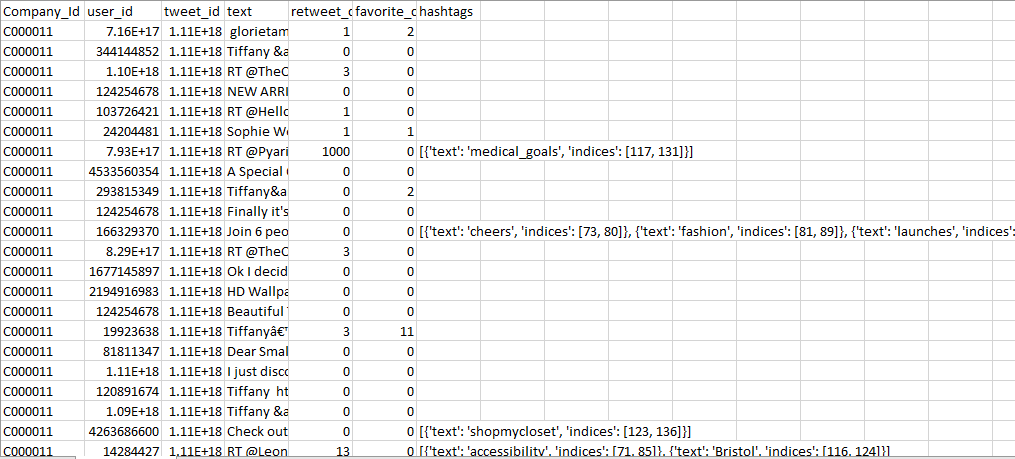
**COMPANY\_ID:** This is the unique id which is given to company.

**TWEET\_TEXT:** Tweet text is the text posted by the user in twitter.

**RETWEET\_COUNT**: Retweet count is the number of people retweeted or replied to any person.

**FAVORITE\_COUNT**: Favorite count is the number of people who liked the particular posts.

**HASHTAG\_TEXT**: Hashtag text is the combination of the hashtags the user posted.



TWITTER\_USER\_DATA\_TBL:

This table has the following attributes and the primary key in the table is USER\_ID which is used to uniquely define each row.

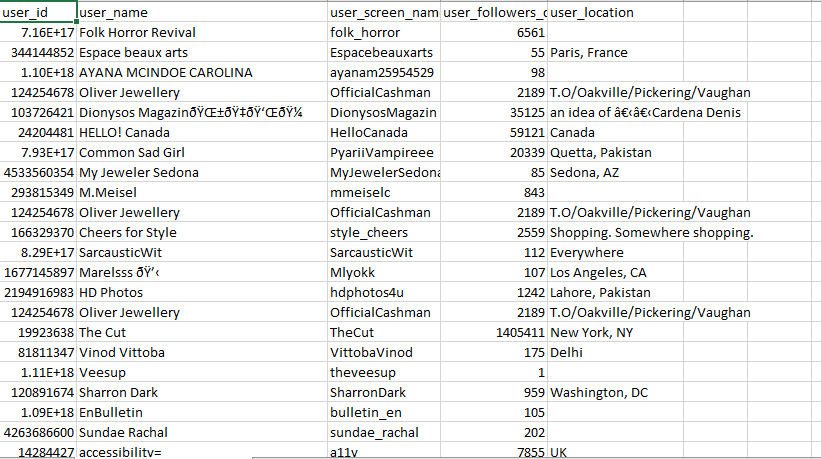
**USER\_ID**: Each person who had a twitter account will be given a unique id. This id is the USER\_ID

**USER\_NAME**: User name is the name of the user who posted the particular tweet.

**USER\_SCREEN\_NAME**: User screen name is the screen name of the user in the twitter.

**USER\_FOLLOWERS**: Number of followers of a particular user.

**USER\_LOCATION**: Location of the user.



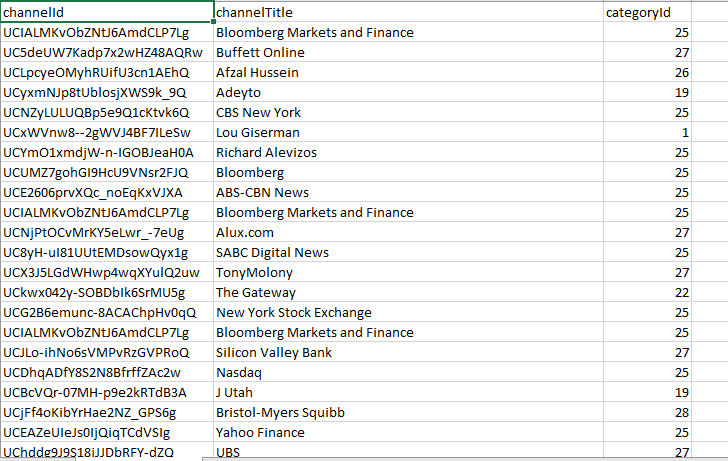
YOUTUBE\_DATA:

This table has the following attributes and the primary key in this table is channel\_id and the foreign key in this table is category\_id which references youtube\_category table.

**CHANNEL\_ID**: Each channel in the YouTube is given the unique id. This unique id is called the CHANNEL\_ID.

**CHANNEL\_TITLE**: Channel\_title is the title of the YouTube channel

**CATEGORY\_ID**: This is the unique id given to each and every category in the category table mainly to eliminate the data redundancy.



YOUTUBE\_CATEGORY:

This table has the following attributes and the primary key in this table is the category\_id.

**CATEGORY\_ID:** This is the unique id given to each category in the category table.

**CATEGORY**: Category gives the information of the category of the video.

**CATEGORY\_DESC**: Description of that specific category. For example, category 24 in YouTube describes the entertainment.



**YOUTUBE\_VIDEO\_DATA**:

This table has the following attributes and the primary key in this table is the VIDEO\_ID and the foreign key in this table are COMPANY\_ID which references COMPANY\_TBL and the CHANNEL\_ID references the YOUTUBE\_DATA table.

**VIDEO\_ID**: VIDEO\_ID represents the unique id given to each video posted in the YouTube.

**COMPANY\_ID**: This is the foreign key which references the COMPANY\_ID in the COMPANY\_TBL.

**VIEW\_COUNT**: VIEW\_COUNT represents the number of views for that table.

**COMMENT\_COUNT**: COMMENT\_COUNT represents the number of counts for that video.

**LIKE\_COUNT**: LIKE\_COUNT represents the number of likes for that video.

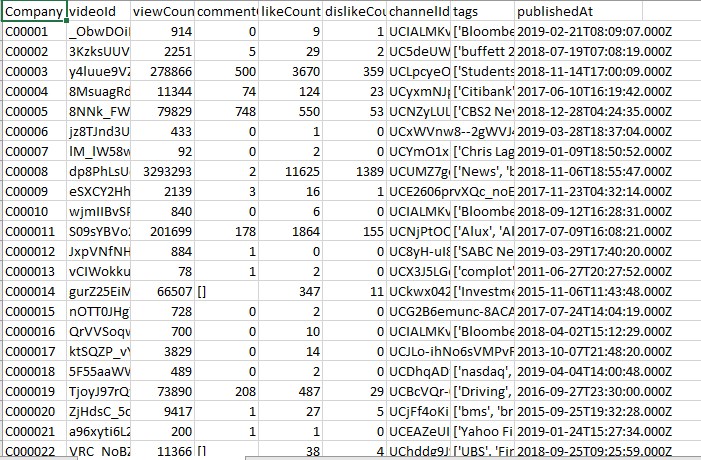
**DISLIKE\_COUNT**: DISLIKE\_COUNT represents the number of dislikes for that video.

**FAVORITE\_COUNT**: FAVORITE\_COUNT represents the number of favorites for that video.

**CHANNEL\_ID**: This is the unique id given to each category in the category table. This is the foreign key in the table.

**TAGS\_CONTENT**: Contents of the tag in the video.

**PUBLISHED\_AT**: The time at which the video is posted or published.



GLASS\_DOOR\_RATINGS

This table has the following attributes and the foreign keys in this table

**COMPANY\_ID**: This is the foreign key which references the COMPANY\_ID in the COMPANY\_TBL.

**RATING\_OVERALL**: It represents the overall Glassdoor rating of that particular company.

**RATING\_BLNC**: It represents the rating of the work life balance for that particular company.

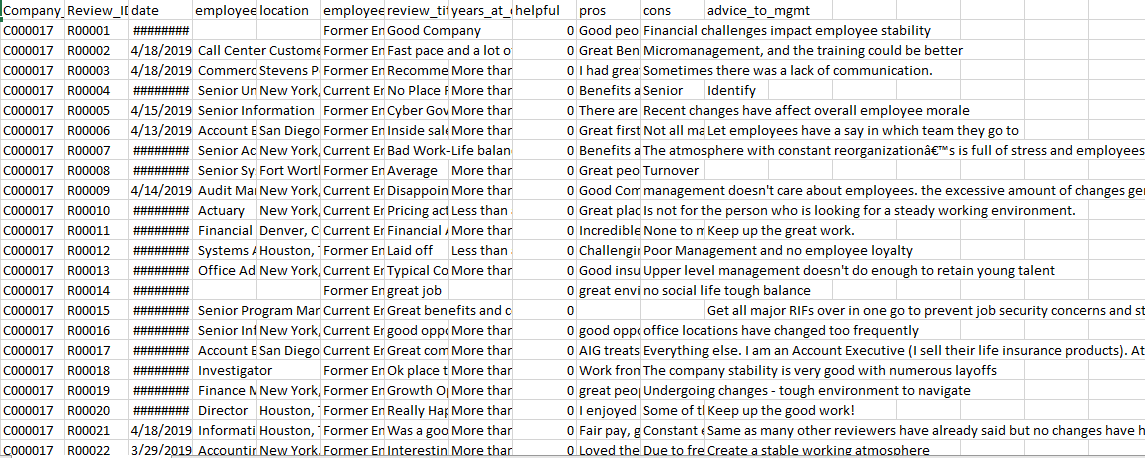
**RATING\_CULTURE**: It represents the culture rating of that company.

**RATING\_CAREER\_GROWTH**: It represents the rating of the career growth i.e how frequently the employer is promoted to the higher positions.

**RATING\_COMP**: It represents the rating for the compensation i.e how well the company pays for the service you have done.

**RATING\_MGMT**: It represents the rating for the management of the company.

**REVIEW\_ID**: It represents the unique id given to the each and every review.



GLASSDOOR\_REVIEWS:

This table has the following attributes:

**COMPANY\_ID**: This is the foreign key which references the COMPANY\_ID in the COMPANY\_TBL.

**DATE**: It give the information about on which date the user posted the review.

**EMPLOYEE\_TITLE**: Employee title is the title in which this person worked and it will be asked while providing the review.

**LOCATION**: The location of the company where he worked.

**EMPLOYEE\_STATUS**: Employee status is the status of the employment i.e he may be a current employee or a Former employee.

**REVIEW\_TITLE**: Title given to each review.

**YEARS\_IN\_COMPANY**: Number of years in worked in that company.

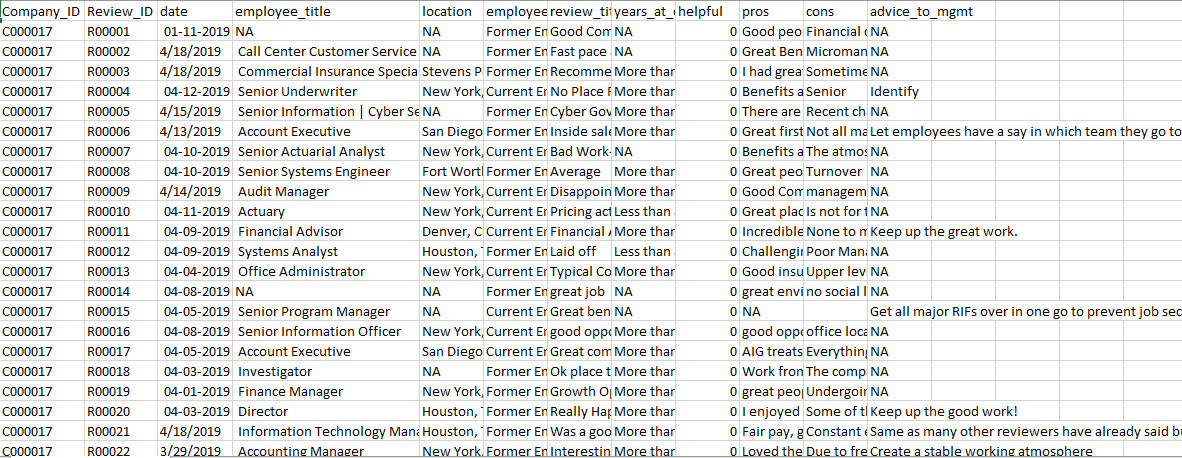
**HELPFUL**: Is this review helpful or not to others who are reading it.

**PROS**: Pros about the company.

**CONS**: Cons about the company

**ADVICE\_TO\_MANAGEMENT**: Any advice prescribed to the management.

**REVIEW\_ID**: It represents the unique id given to the each and every review.



**NORMALIZATION:**

After the tables are created then the next step is data normalization. Normalization is used to reduce the data redundancy. We can’t eliminate the data redundancy completely, but we can reduce the redundancy by dividing the repeating columns in the particular table into a new table and generate a unique Id to that table. Now instead of repeating of all the columns we will give this unique id to the table and it acts a s link between them.

**1ST NORMALIZATION FORM:**

A table in 1NF should be atomic and have non repeating rows and columns.

Our tables are in 1NF as they satisfy each requirement of first Normalization form.

**2ND NORMALIZATION FORM:**

There should not be any partial dependency, which means that no value in the table should be dependent on a part of primary key.

Our tables are in 2NF as they satisfy every requirement of second Normalization form.

**3RD NORMALIZATION FORM:**

A table is said to be in 3NF if no non primary attribute in the table should be dependent on other nonprimary attribute in the table.

Our tables are in 3NF as they satisfy every requirement of third Normalization form.

**USES CASES:**

#### **Use case-1**: Select company with most job postings

Code:

SELECT B.COMPANY\_NAME, COUNT(A.JOB\_LOCATION) AS POSITION\_COUNT FROM JOBS\_TBL A

JOIN COMPANY\_TBL B

ON A.COMPANY\_ID = B.COMPANY\_ID

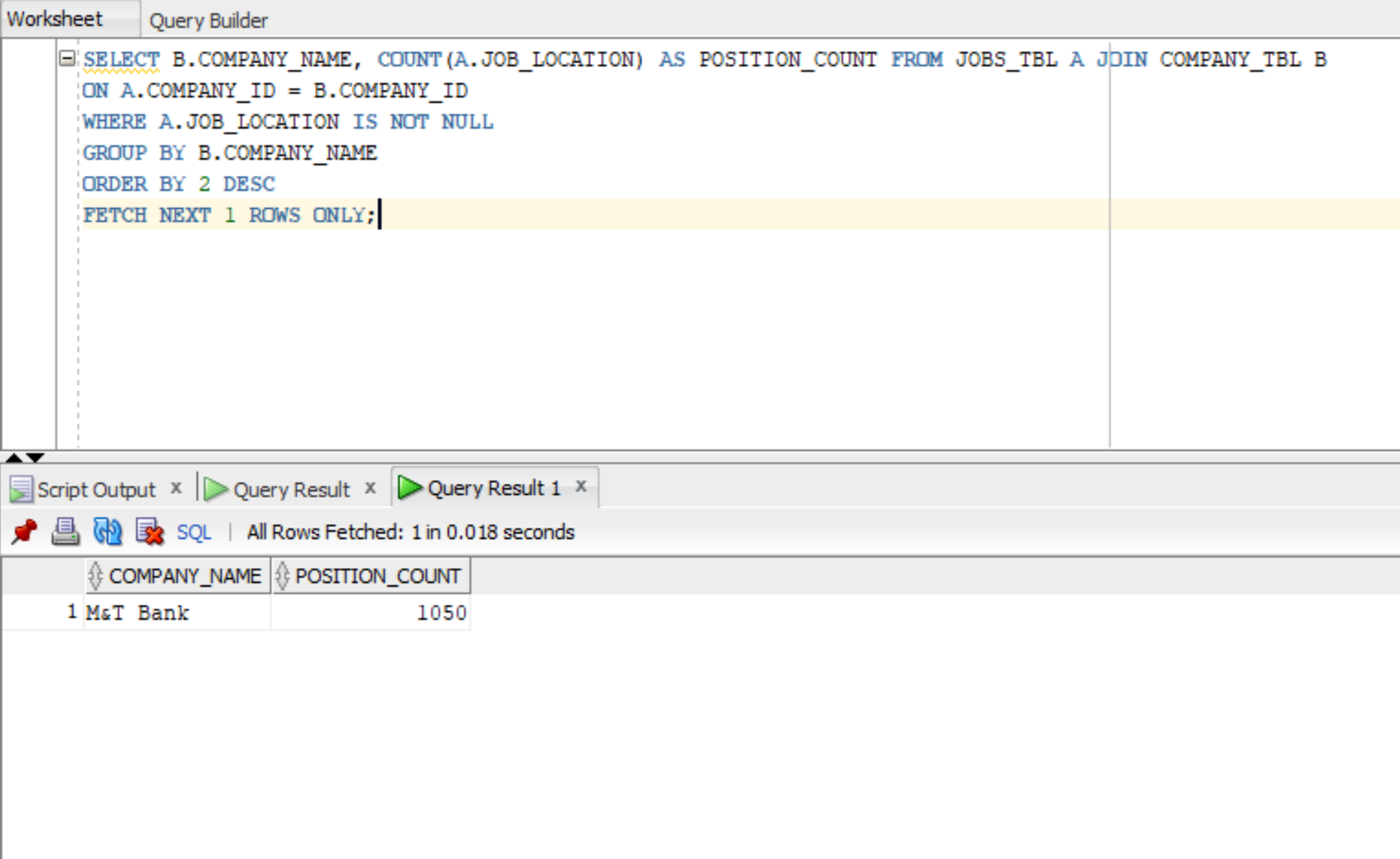
WHERE A.JOB\_LOCATION IS NOT NULL

GROUP BY B. COMPANY\_NAME

ORDER BY 2 DESC

FETCH NEXT 1 ROWS ONLY;

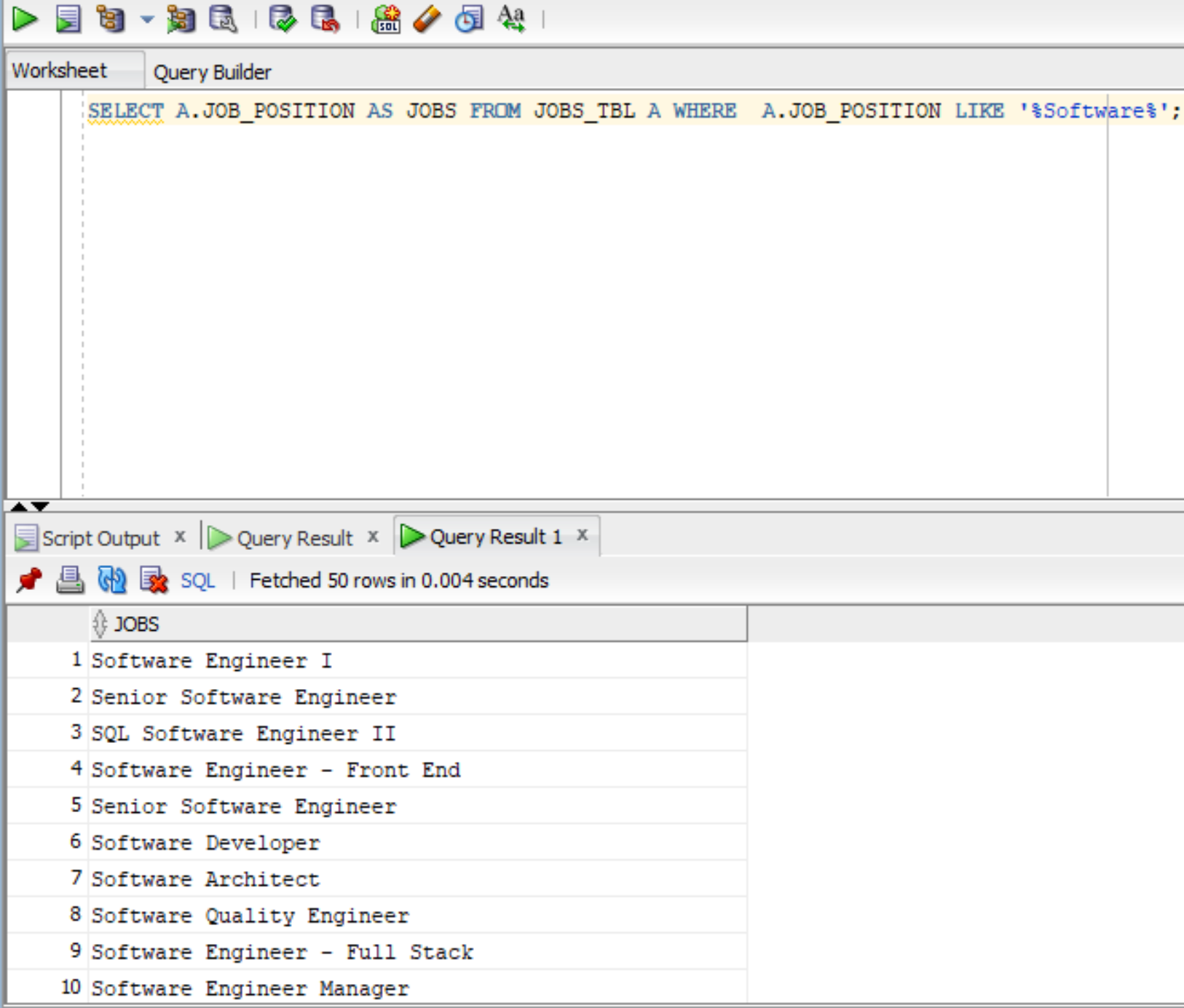
**Output**:



#### **Use case-2**: Get a List of 'Software Engineering' Jobs

Code: SELECT A.JOB\_POSITION AS JOBS FROM JOBS\_TBL A WHERE A.JOB\_POSITION LIKE '%Software%';

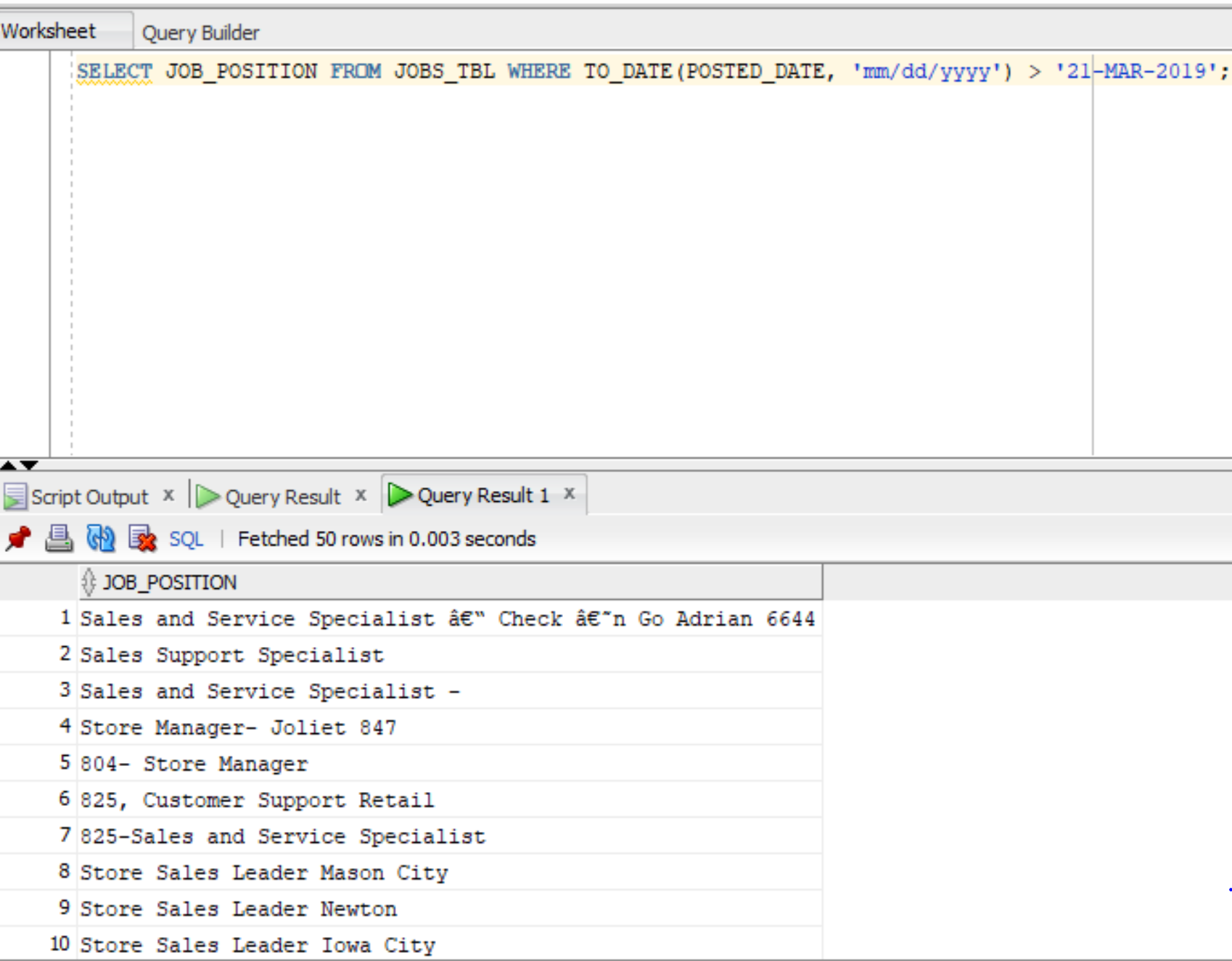
Output:



#### **Use case-3**: Get a List of most recent postings Jobs

Code: SELECT JOB\_POSITION FROM JOBS\_TBL WHERE TO\_DATE(POSTED\_DATE, 'mm/dd/yyyy') > '21-MAR-2019';

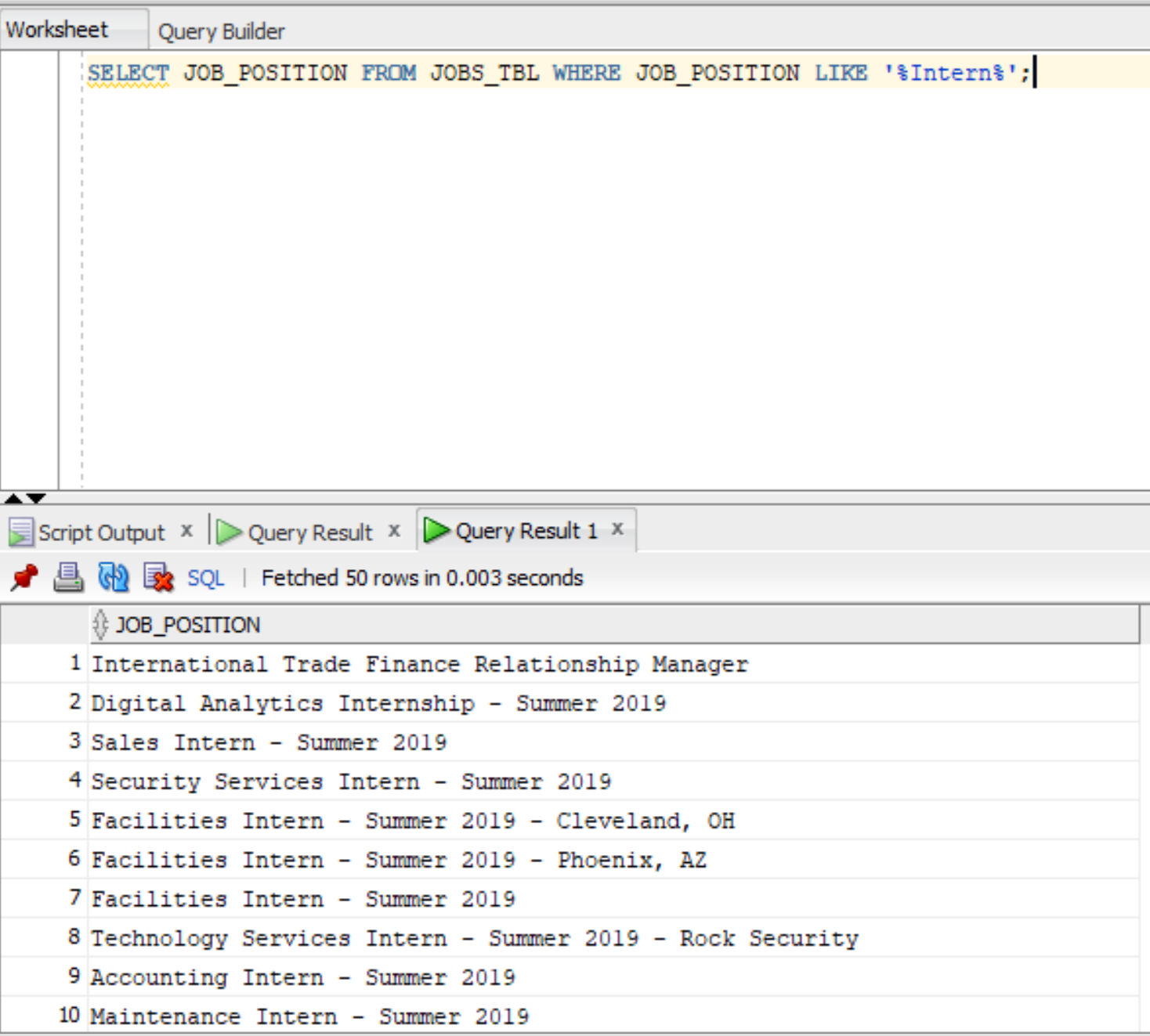
Output:



#### Use case-4: Get a list of all the internships

Code:

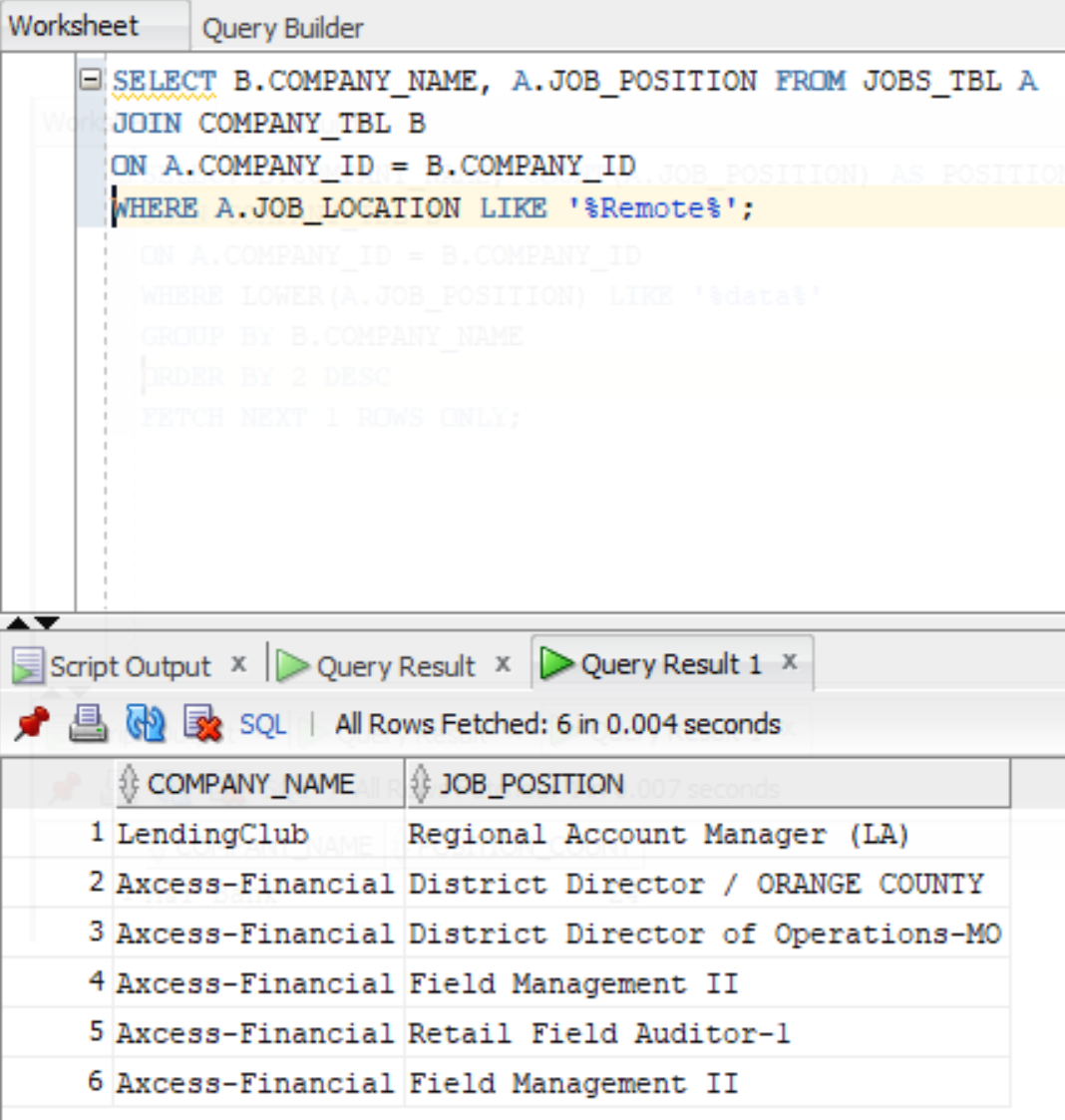
Output:



#### Use case-5: Get a list of all the companies with remote jobs

Code: SELECT B.COMPANY\_NAME, A.JOB\_POSITION FROM JOBS\_TBL A JOIN COMPANY\_TBL B ON A.COMPANY\_ID = B.COMPANY\_ID WHERE A.JOB\_LOCATION LIKE '%Remote%';

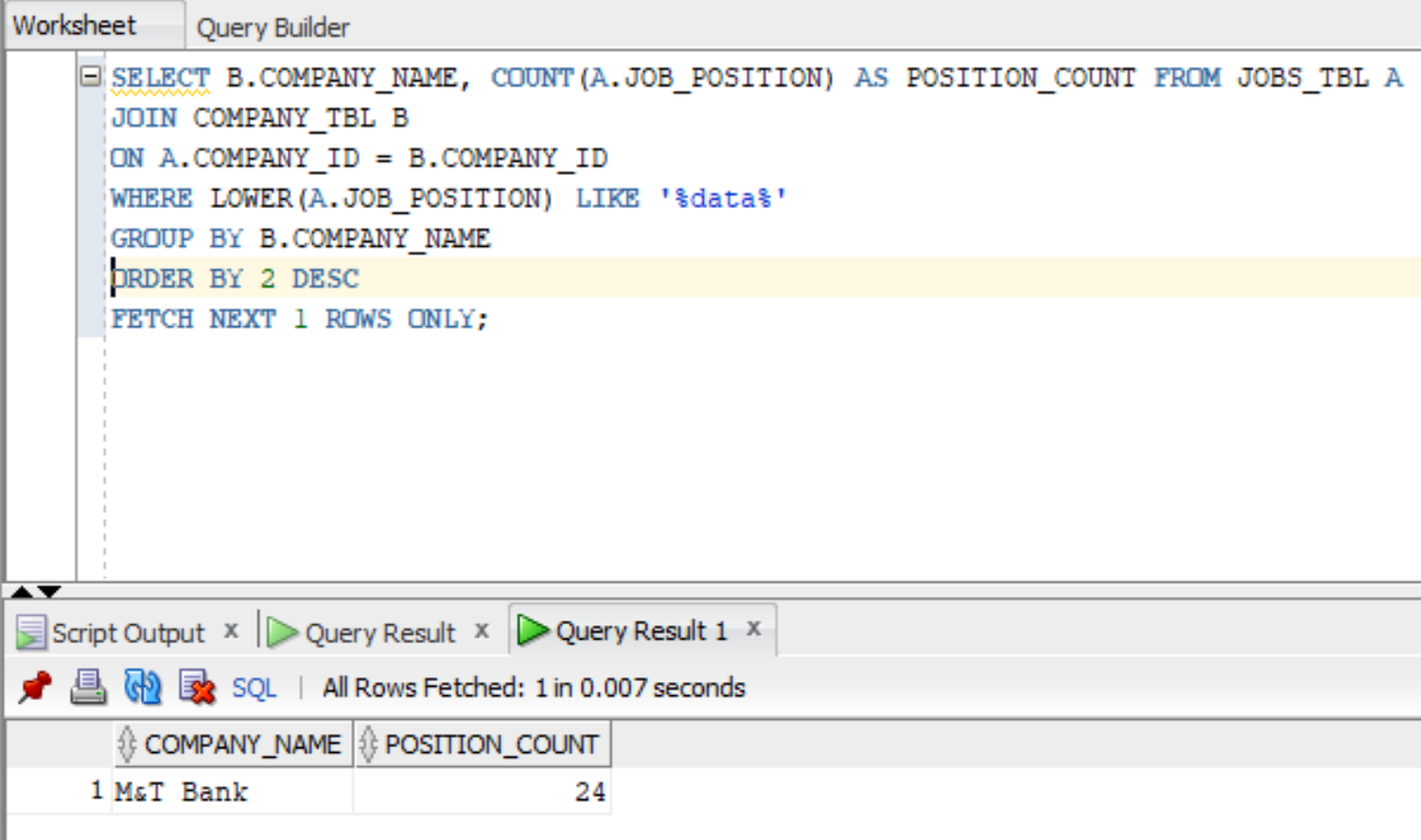
Output:



#### Use case-6: Get the name of the company with more data related jobs

Code: SELECT B.COMPANY\_NAME, COUNT(A.JOB\_POSITION) AS POSITION\_COUNT FROM JOBS\_TBL A JOIN COMPANY\_TBL B ON A.COMPANY\_ID = B.COMPANY\_ID WHERE LOWER(A.JOB\_POSITION) LIKE '%data%' GROUP BY B.COMPANY\_NAME ORDER BY 2 DESC FETCH NEXT 1 ROWS ONLY;

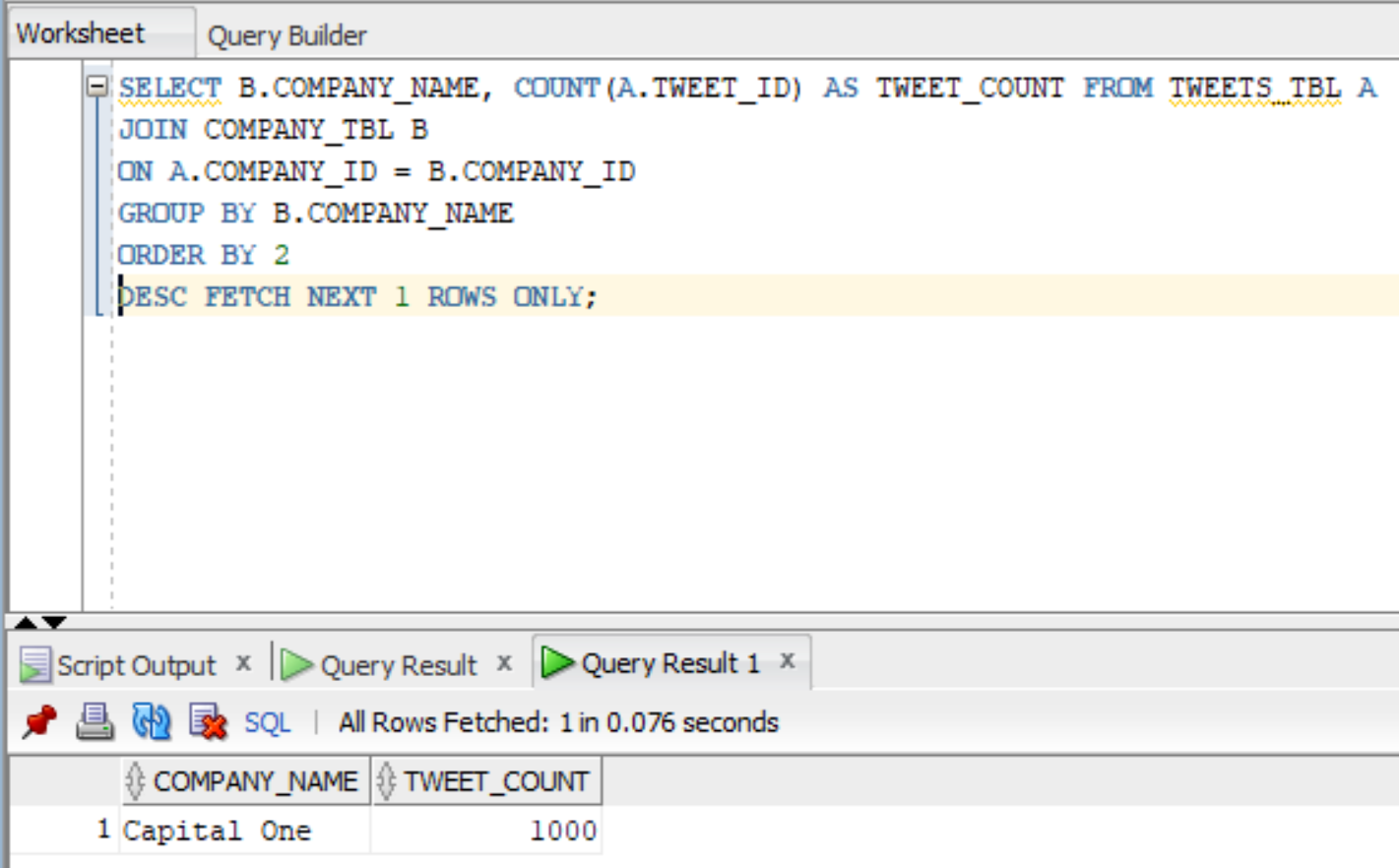
Output:



#### Use case-7: Get the name of the company which is active on twitter, the company with most tweets

Code: SELECT B.COMPANY\_NAME, COUNT(A.TWEET\_ID) AS TWEET\_COUNT FROM TWEETS\_TBL A JOIN COMPANY\_TBL B ON A.COMPANY\_ID = B.COMPANY\_ID GROUP BY B.COMPANY\_NAME ORDER BY 2 DESC FETCH NEXT 1 ROWS ONLY;

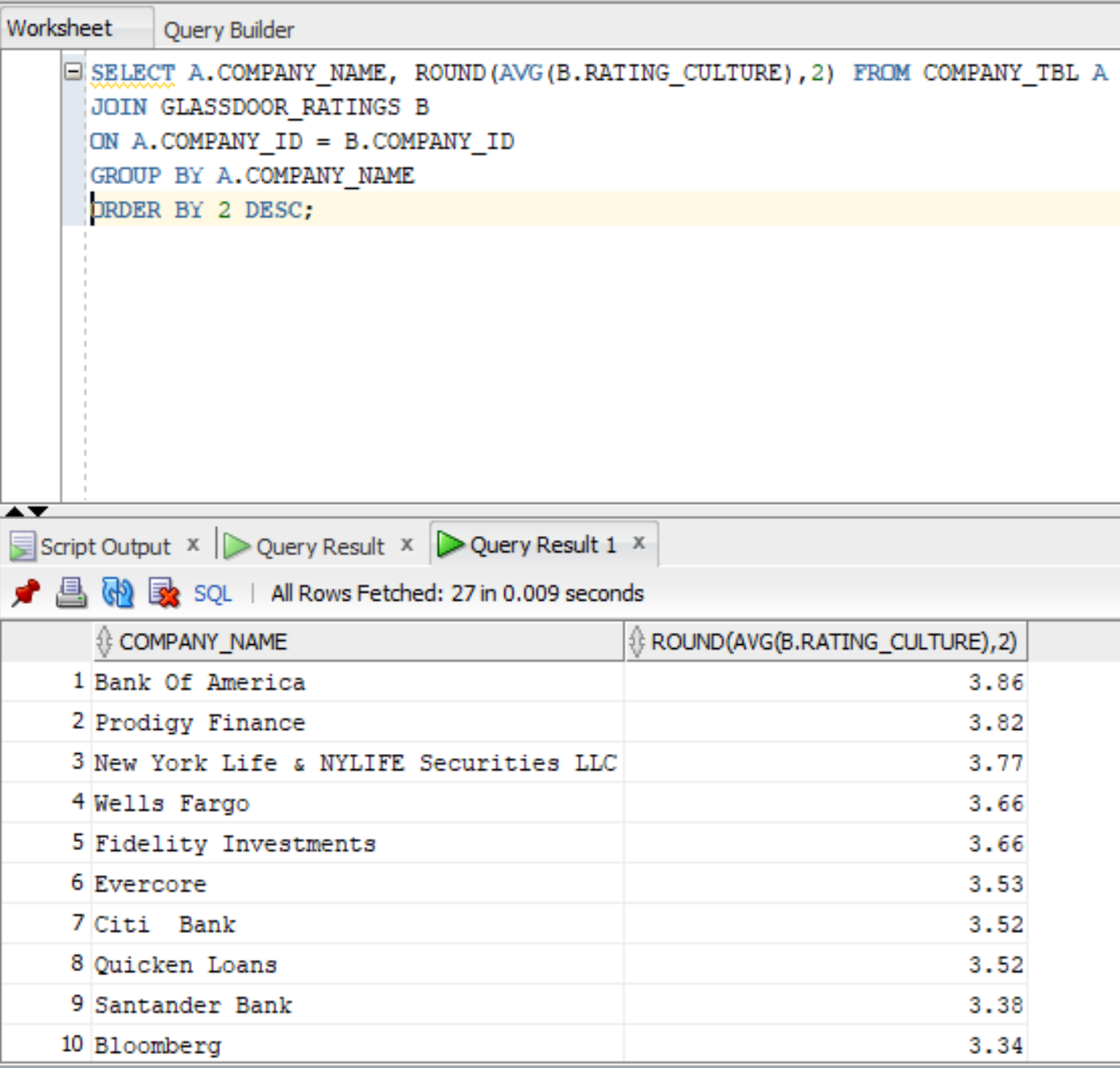
Output:



#### Use case-8:  Get the AVERAGE rating\_culture of the companies

Code: SELECT A.COMPANY\_NAME, ROUND(AVG(B.RATING\_CULTURE),2) FROM COMPANY\_TBL A JOIN GLASSDOOR\_RATINGS B ON A.COMPANY\_ID = B.COMPANY\_ID GROUP BY A.COMPANY\_NAME ORDER BY 2 DESC;

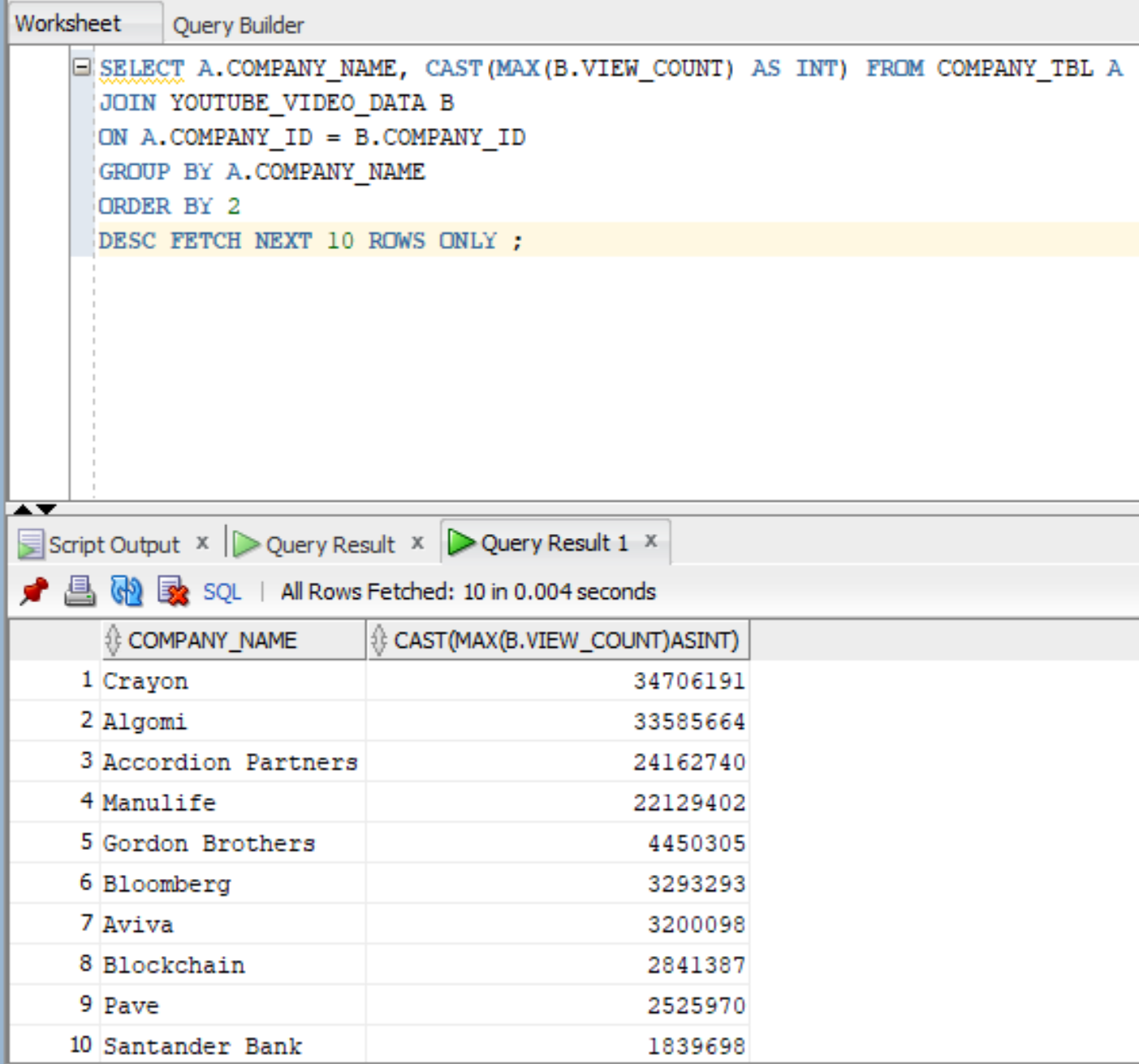
Output:



#### Use case-9: Get the names of the companies with the most YouTube video views[¶](http://localhost:8890/notebooks/OneDrive/Desktop/DMDD/Final%20Project/10%20SQL%20USECASES-FUNC_VIEWS.ipynb#UseCase-9---Get-the-names-of-the-companies-with-the-most-youtube-video-views)

Code: SELECT A.COMPANY\_NAME, CAST(MAX(B.VIEW\_COUNT) AS INT) FROM COMPANY\_TBL A JOIN YOUTUBE\_VIDEO\_DATA B ON A.COMPANY\_ID = B.COMPANY\_ID GROUP BY A.COMPANY\_NAME ORDER BY 2 DESC FETCH NEXT 10 ROWS ONLY ;

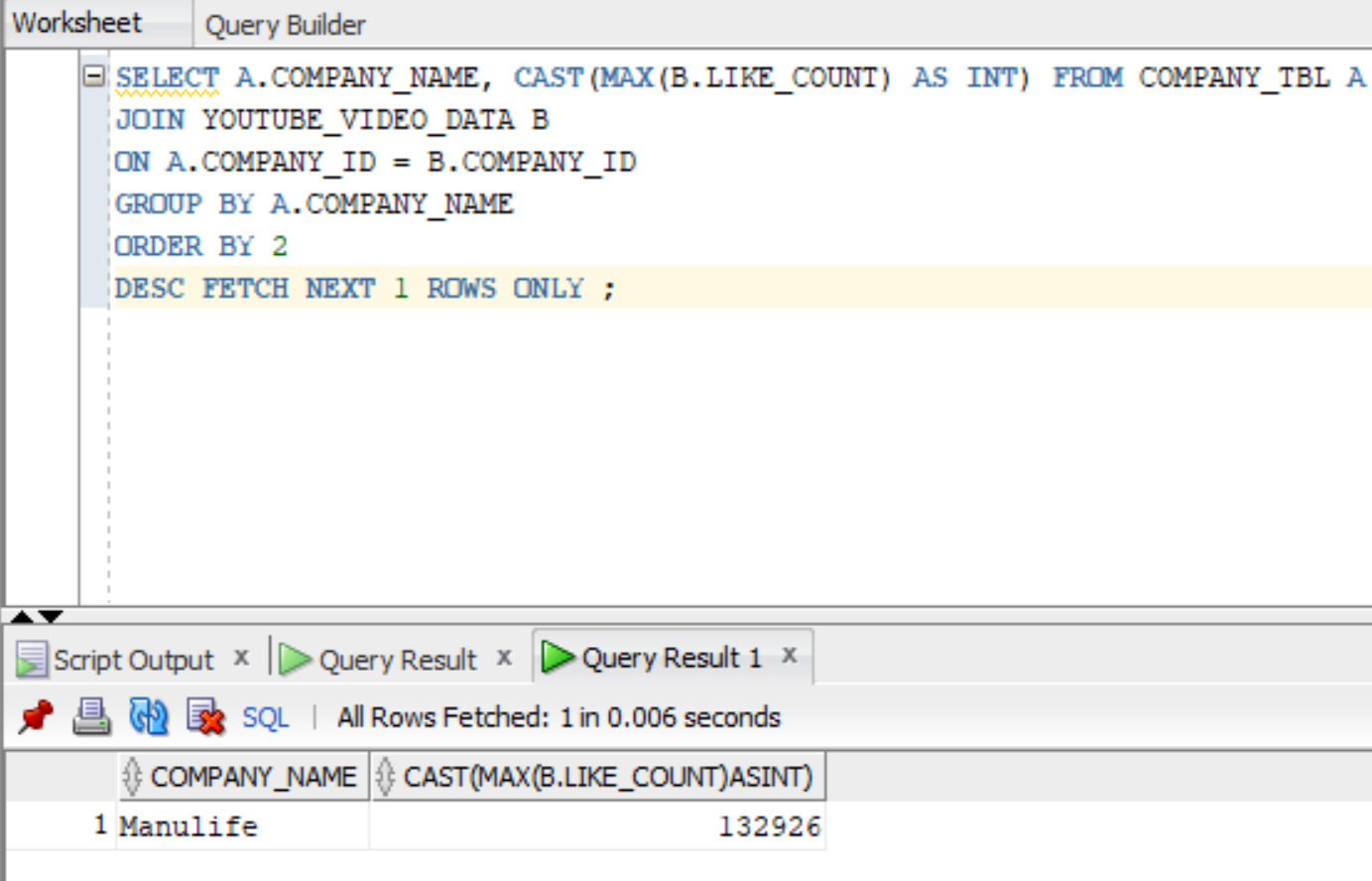
Output:



#### Use case-10: Get the name of the company with more video likes

Code: SELECT A.COMPANY\_NAME, CAST(MAX(B.LIKE\_COUNT) AS INT) FROM COMPANY\_TBL A JOIN YOUTUBE\_VIDEO\_DATA B ON A.COMPANY\_ID = B.COMPANY\_ID GROUP BY A.COMPANY\_NAME ORDER BY 2 DESC FETCH NEXT 1 ROWS ONLY ;

Output:

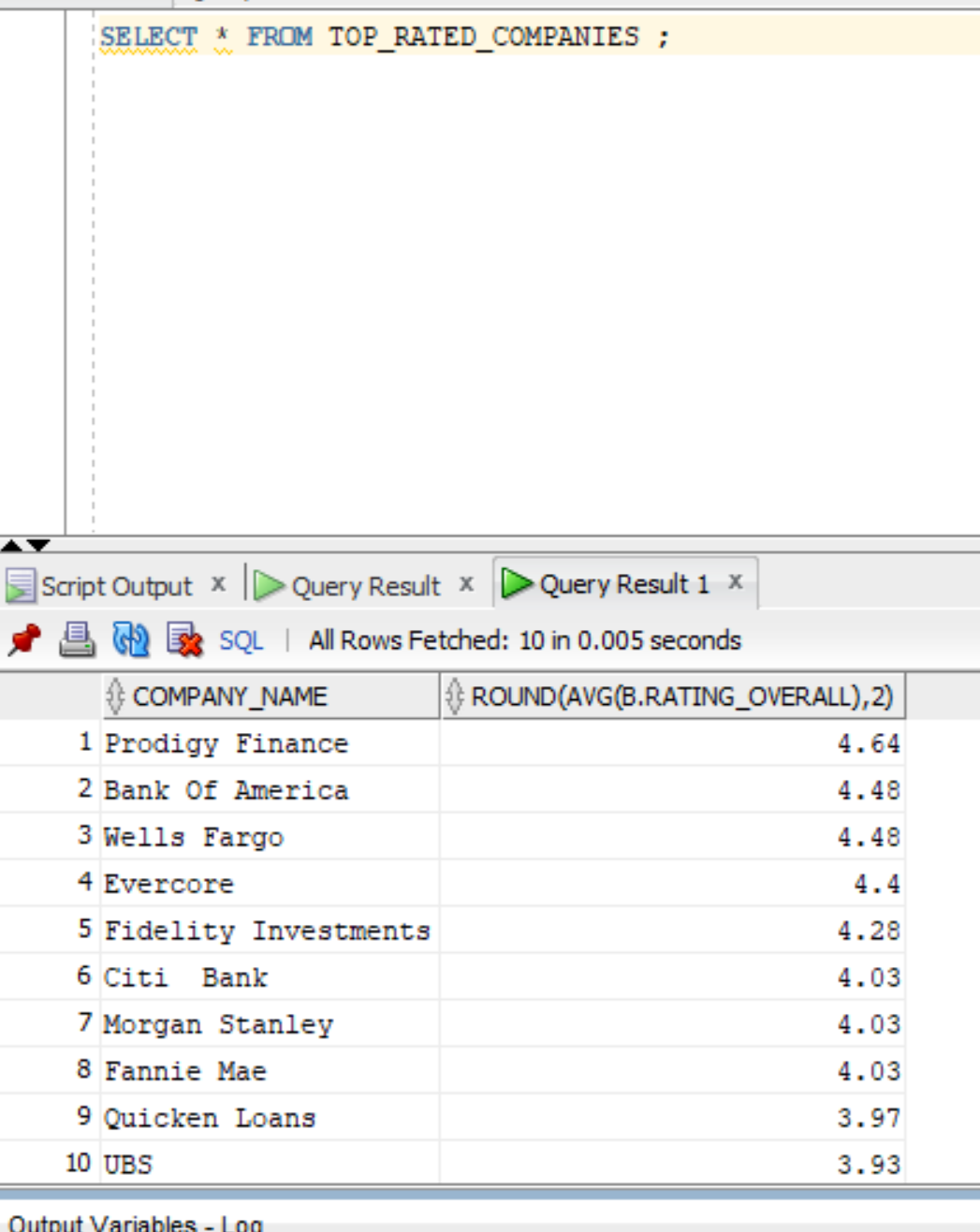


**VIEWS**:

#### **View-1**:  Get the TOP 10 rated companies and their average glass door ratings

Code: CREATE VIEW TOP\_RATED\_COMPANIES AS SELECT A.COMPANY\_NAME, ROUND(AVG(B.RATING\_OVERALL),2) FROM COMPANY\_TBL A JOIN GLASSDOOR\_RATINGS B ON A.COMPANY\_ID = B.COMPANY\_ID GROUP BY A.COMPANY\_NAME ORDER BY 2 DESC FETCH NEXT 10 ROWS ONLY;

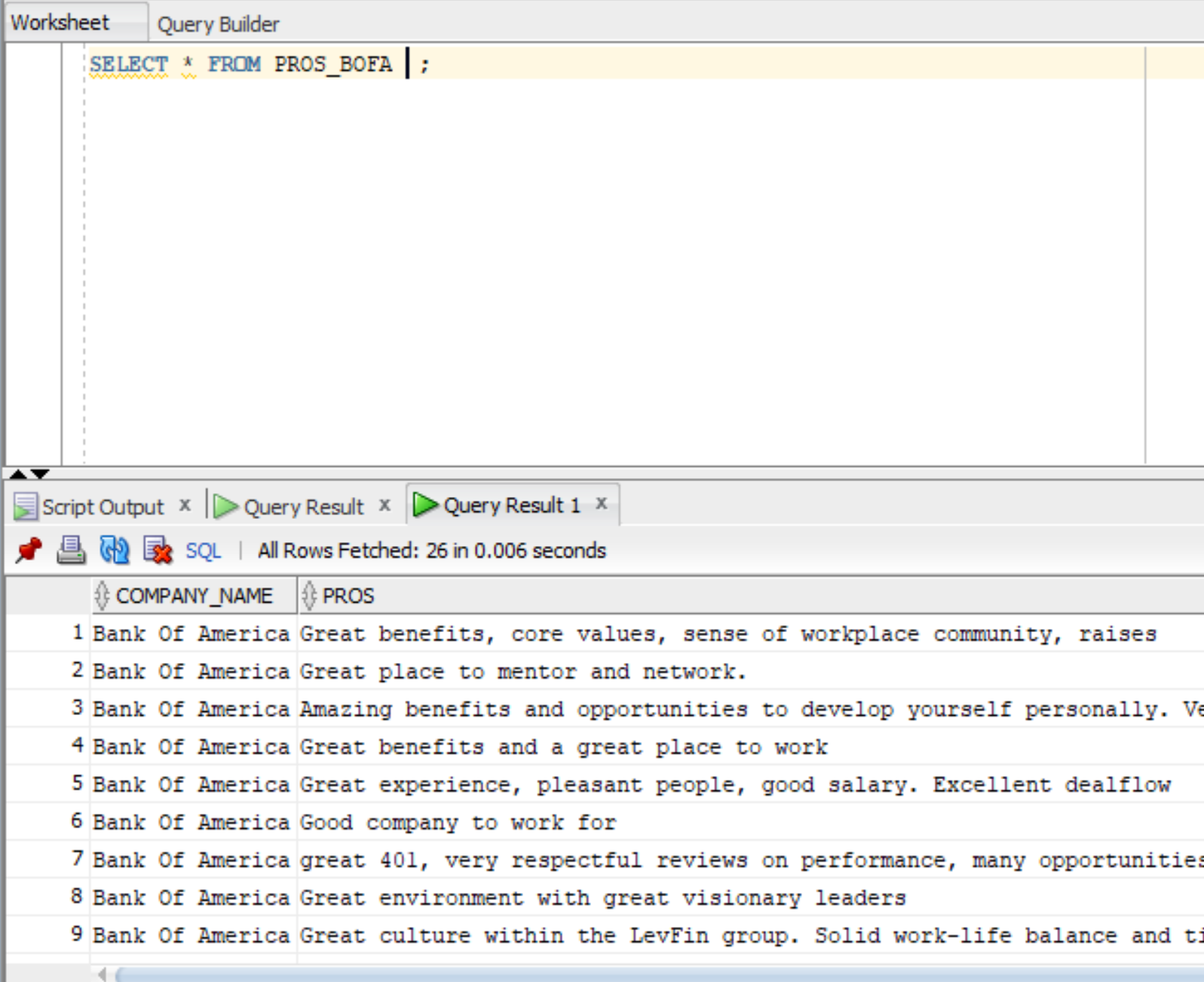
Output:



#### **View-2**: Get the company and their glass door pros - BOFA

Code: CREATE VIEW PROS\_BOFA AS SELECT A.COMPANY\_NAME, B.PROS FROM COMPANY\_TBL A JOIN GLASSDOOR\_REVIEWS B ON A.COMPANY\_ID = B.COMPANY\_ID AND A.COMPANY\_NAME = 'Bank Of America';

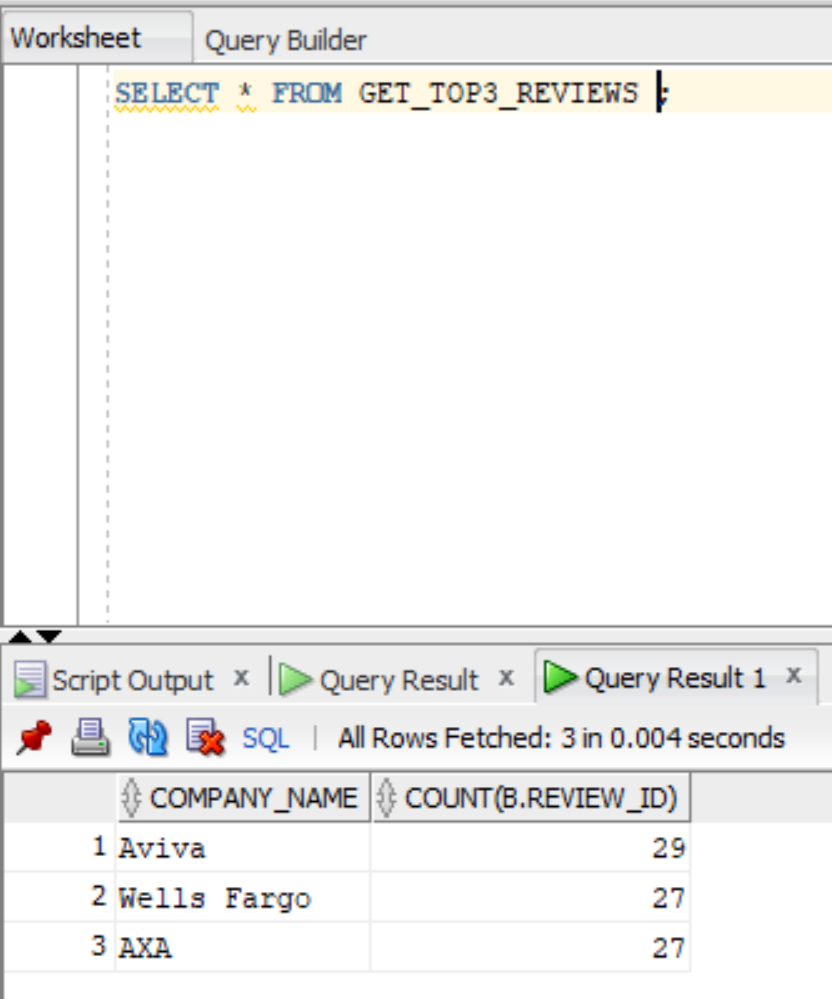
Output:



#### **View-3**: Get the TOP 3 company with more reviews

Code: CREATE VIEW GET\_TOP3\_REVIEWS AS SELECT A.COMPANY\_NAME, COUNT(B.REVIEW\_ID) FROM COMPANY\_TBL A JOIN GLASSDOOR\_REVIEWS B ON A.COMPANY\_ID = B.COMPANY\_ID GROUP BY A.COMPANY\_NAME ORDER BY 2 DESC FETCH NEXT 3 ROWS ONLY;

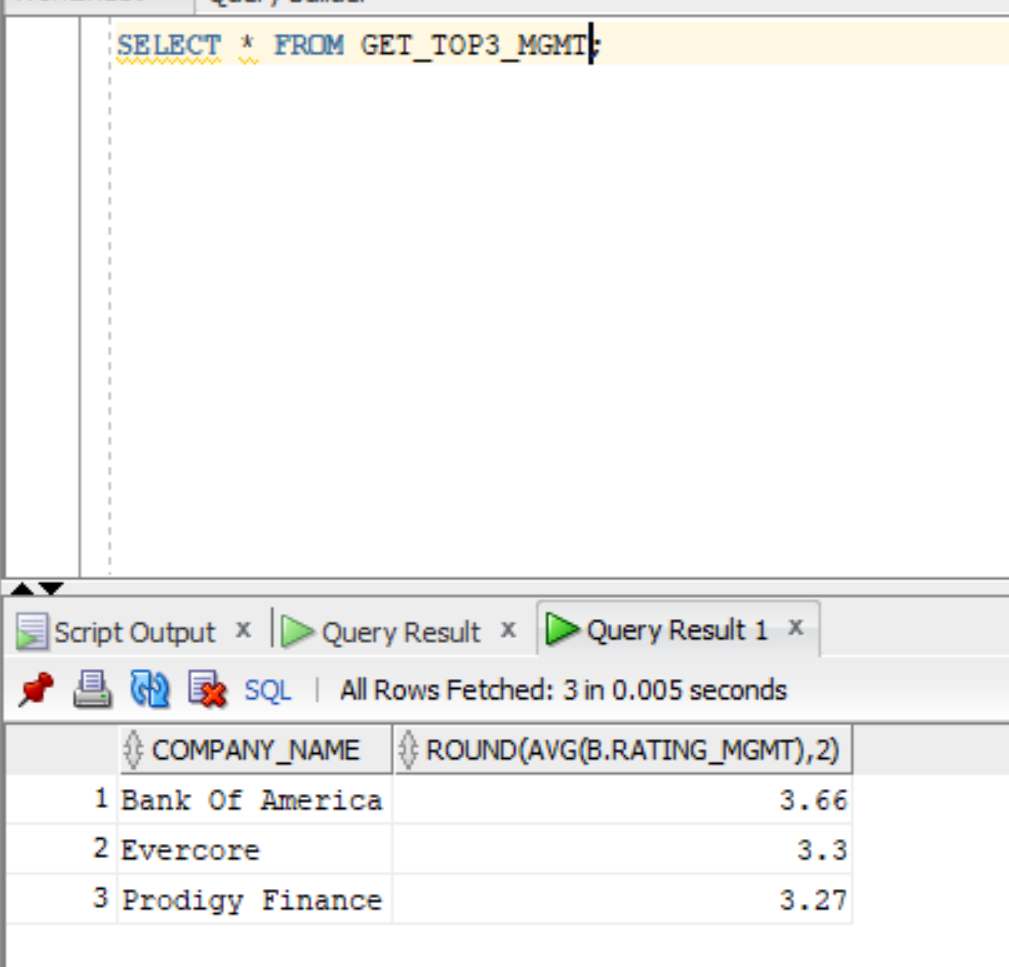
Output:



#### **View-4**: Get the company and their glass door ratings

Code: CREATE VIEW GET\_TOP3\_MGMT AS SELECT A.COMPANY\_NAME, ROUND(AVG(B.RATING\_MGMT),2) FROM COMPANY\_TBL A JOIN GLASSDOOR\_RATINGS B ON A.COMPANY\_ID = B.COMPANY\_ID GROUP BY A.COMPANY\_NAME ORDER BY 2 DESC FETCH NEXT 3 ROWS ONLY ;

Output:

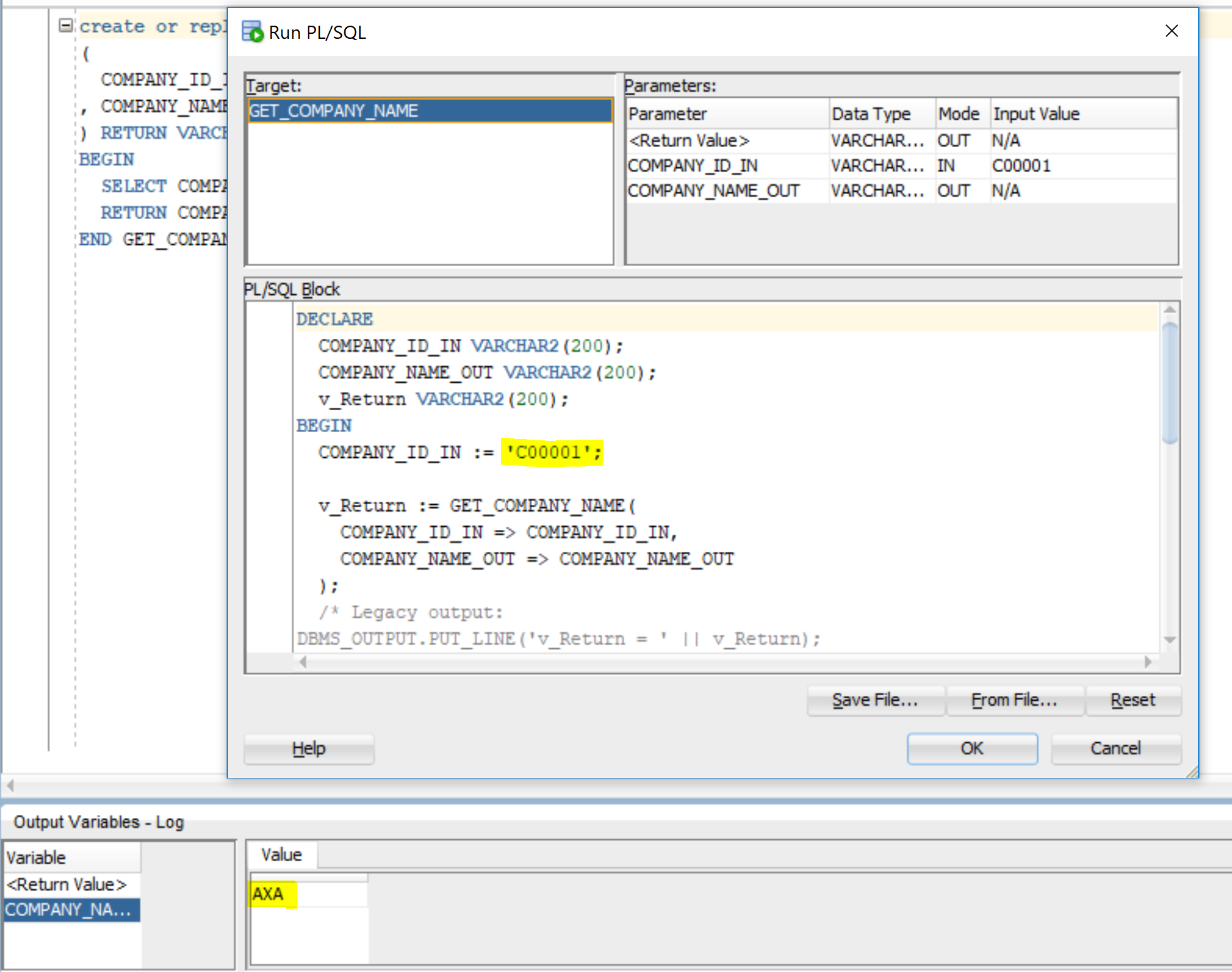


**FUNCTIONS**:

#### Function-1: Get the name of the company given COMPANY\_ID

Code: CREATE OR REPLACE FUNCTION GET\_COMPANY\_NAME ( COMPANY\_ID\_IN IN VARCHAR2 , COMPANY\_NAME\_OUT OUT VARCHAR2 ) RETURN VARCHAR2 AS BEGIN SELECT COMPANY\_NAME INTO COMPANY\_NAME\_OUT FROM COMPANY\_TBL WHERE COMPANY\_ID = COMPANY\_ID\_IN ; RETURN COMPANY\_NAME\_OUT; END GET\_COMPANY\_NAME;

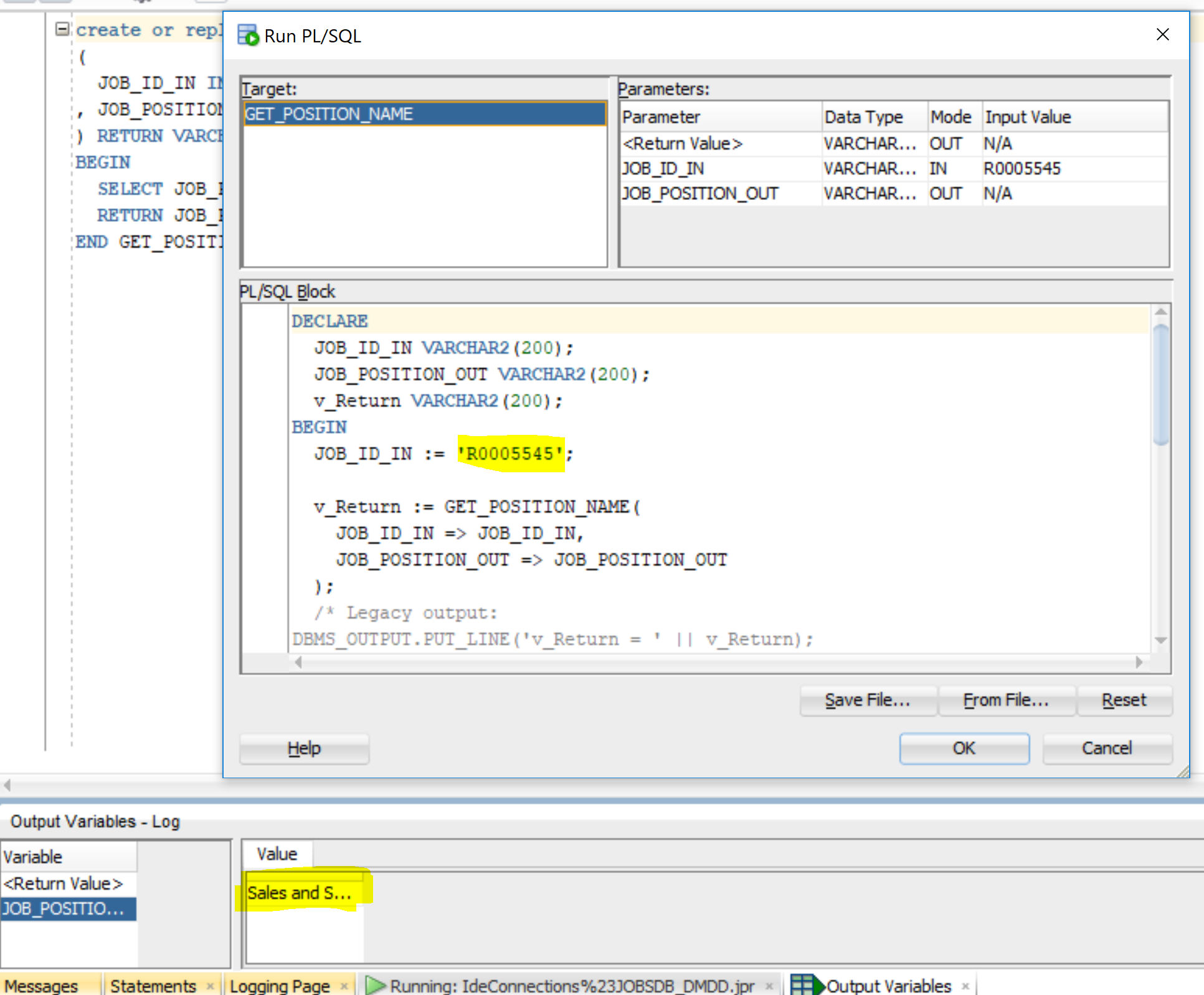
Output:



#### Function-2: Get the count of user followers given user\_id - ex - 798908432

Code: CREATE OR REPLACE FUNCTION GET\_USER\_FOLLOWERS ( USER\_ID\_INPUT IN VARCHAR2 , USER\_FOLLOWERS\_OUTPUT OUT VARCHAR2 ) RETURN VARCHAR2 AS BEGIN SELECT USER\_FOLLOWERS INTO USER\_FOLLOWERS\_OUTPUT FROM TWITTER\_USER\_DATA\_TBL WHERE USER\_ID = USER\_ID\_INPUT; RETURN USER\_FOLLOWERS\_OUTPUT; END GET\_USER\_FOLLOWERS;

Output:



#### Function-3:  Get the position name given the JOB\_ID

Code: CREATE OR REPLACE FUNCTION GET\_POSITION\_NAME ( JOB\_ID\_IN IN VARCHAR2 , JOB\_POSITION\_OUT OUT VARCHAR2 ) RETURN VARCHAR2 AS BEGIN SELECT JOB\_POSITION INTO JOB\_POSITION\_OUT FROM JOBS\_TBL WHERE JOB\_ID = JOB\_ID\_IN; RETURN JOB\_POSITION\_OUT; END GET\_POSITION\_NAME;

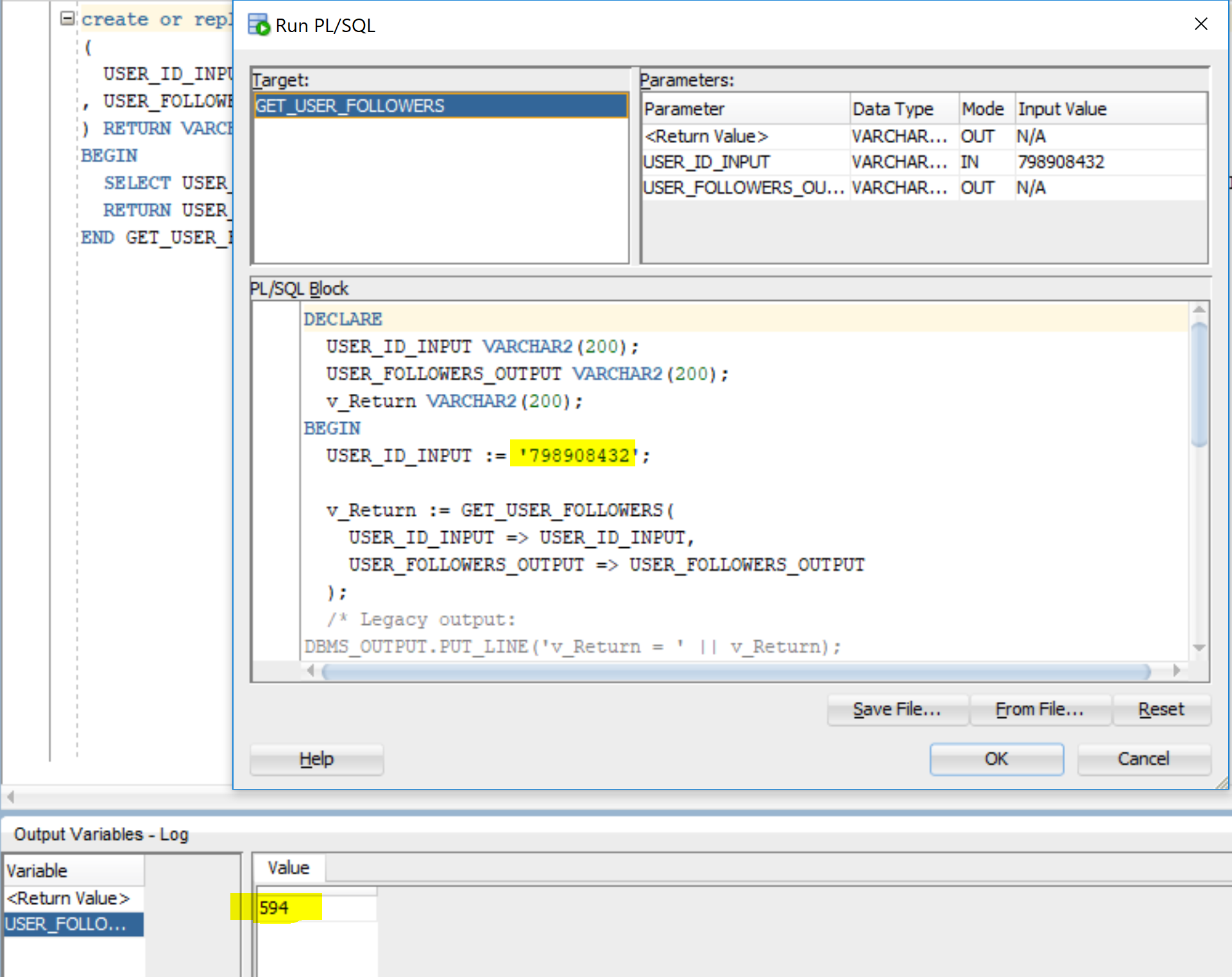
Output:



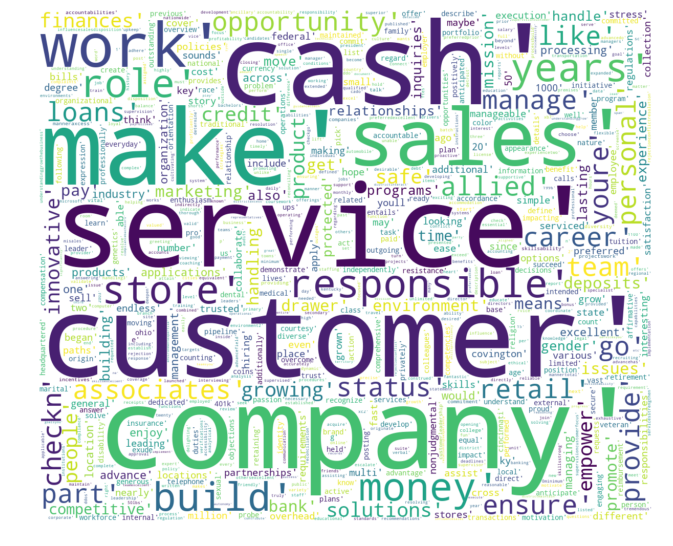
#### Function-4: Get the URL of the job given job\_id

Code: CREATE OR REPLACE FUNCTION GET\_URL ( JOB\_ID\_IN IN VARCHAR2 , JOB\_URL\_OUT OUT VARCHAR2 ) RETURN VARCHAR2 AS BEGIN SELECT JOB\_URL INTO JOB\_URL\_OUT FROM JOBS\_TBL WHERE JOB\_ID = JOB\_ID\_IN; RETURN JOB\_URL\_OUT; END GET\_URL;

Output:



WORD CLOUD:



We can infer from the word cloud that most of the fintech companies focus on customer service. Also you can see finance related terms like cash, money, checkings, deposit etc.