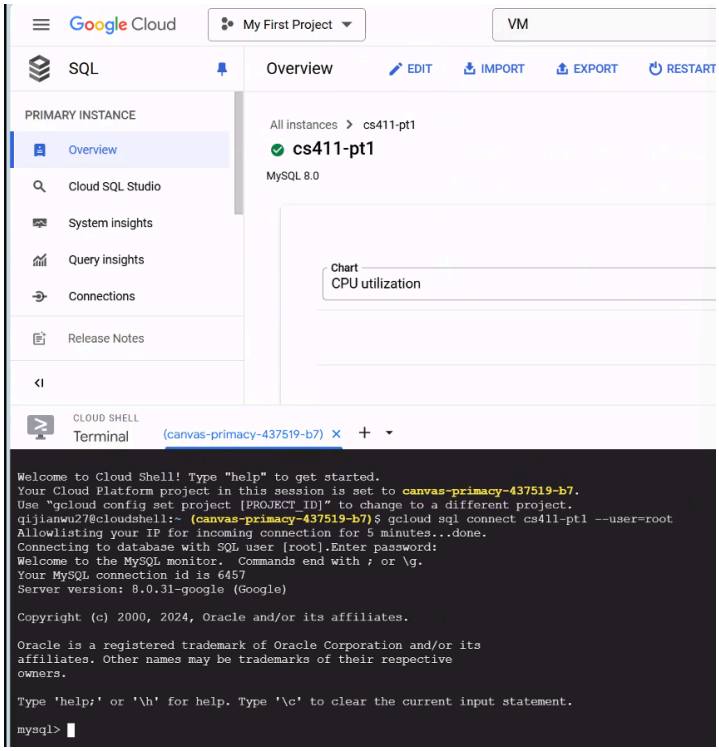
Part1:

Database Connection Screenshot:



Data Definition Language (DDL) commands:

```
CREATE TABLE User (
    UserID VARCHAR(50) PRIMARY KEY,
    UserName VARCHAR(255) NOT NULL,
    Password VARCHAR(50) NOT NULL,
    is_Admin BOOLEAN DEFAULT FALSE,
    FirstName VARCHAR(255),
    LastName VARCHAR(255),
    Age INT,
    Location VARCHAR(50),
    PhoneNumber VARCHAR(50),
    EmailAddress VARCHAR(255)
);
CREATE TABLE Job (
    JobID VARCHAR(255) PRIMARY KEY,
    JobTitle VARCHAR(255),
    JobSnippet VARCHAR(4096),
```

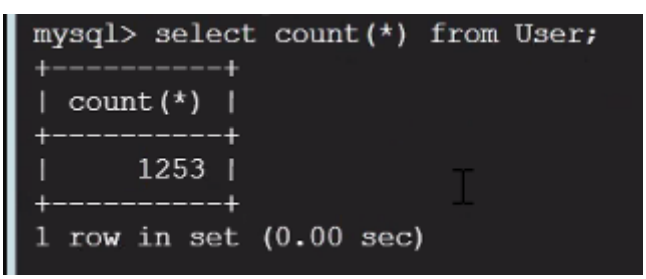
```
    JobLink VARCHAR(255),
    Sponsored BOOLEAN,
    Salary INT,
    Rating REAL,
    CompanyName VARCHAR(255),
    ApprovalStatus BOOLEAN,
    FOREIGN KEY (CompanyName) REFERENCES Company(CompanyName)
);
CREATE TABLE Company (
    CompanyName VARCHAR(255) PRIMARY KEY,
    Location VARCHAR(255),
    CompanyOverview VARCHAR(4096)
);
CREATE TABLE Review (
    ReviewID VARCHAR(50) PRIMARY KEY,
    JobID VARCHAR(255),
    Content VARCHAR(4096),
    Rating REAL,
    FOREIGN KEY (JobID) REFERENCES Job(JobID)
);
CREATE TABLE UploadedHistory (
    UploadID VARCHAR(50) PRIMARY KEY,
    UserID VARCHAR(50),
    JobID VARCHAR(255),
    AdminComment VARCHAR(512),
    FOREIGN KEY (UserID) REFERENCES User(UserID),
    FOREIGN KEY (JobID) REFERENCES Job(JobID)
);
CREATE TABLE Favorite (
    JobID VARCHAR(255),
    UserID VARCHAR(50),
    PRIMARY KEY (JobID, UserID),
```

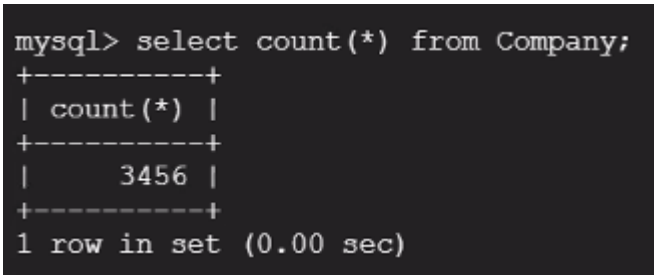
FOREIGN KEY (JobID) REFERENCES Job(JobID),

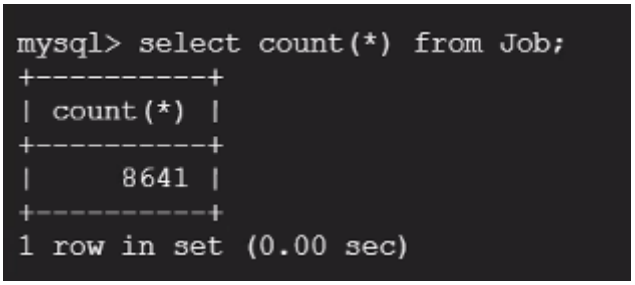
FOREIGN KEY (UserID) REFERENCES User(UserID)

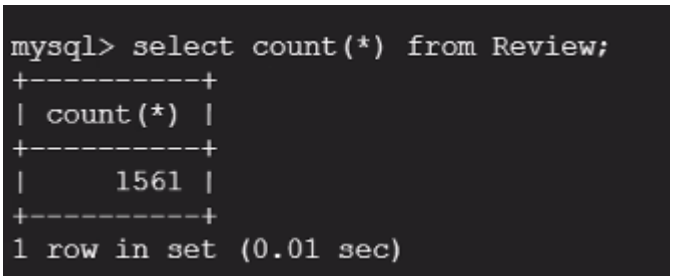
);

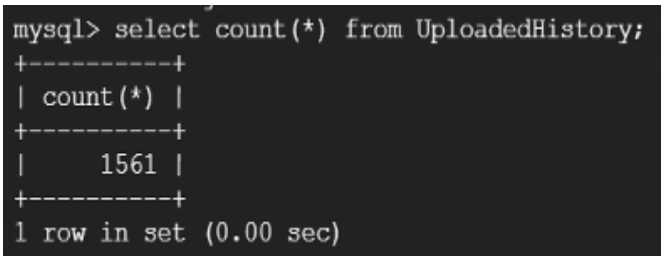
Screenshot of tables:

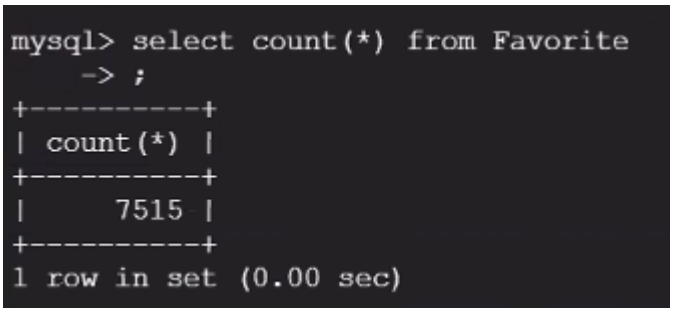








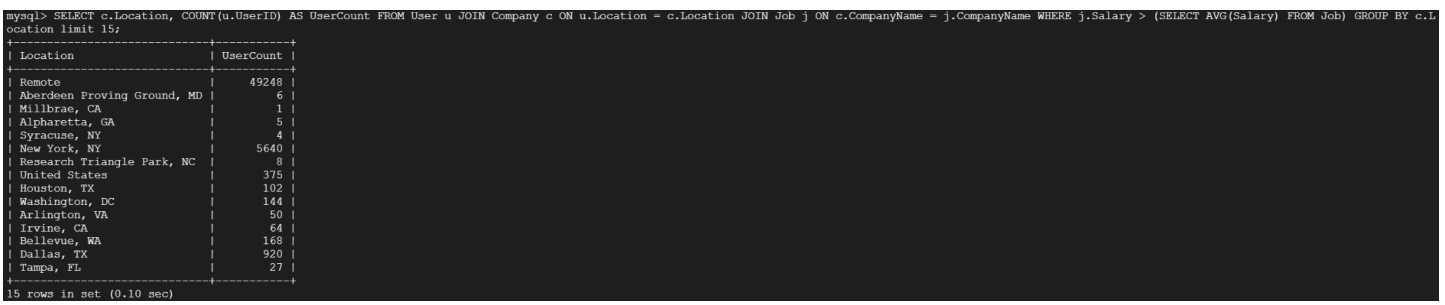




Advanced queries:

-- Query 1: Job Market Analysis with Reviews
-- Purpose: Shows job distribution, salaries, and ratings by location (Count the number of users per location where there are jobs whose salaries are higher than average.)
-- Uses: Multiple JOINs (three relations) and GROUP BY aggregation

SELECT c.Location, COUNT(u.UserID) AS UserCount FROM User u JOIN Company c ON u.Location = c.Location JOIN Job j ON c.CompanyName = j.CompanyName WHERE j.Salary > (SELECT AVG(Salary) FROM Job) GROUP BY c.Location limit 15;



-- Query 2: Salary Analytics with Reviews
-- Purpose: Compare salaries and ratings across locations and job titles

```
-- Uses: Multiple JOINs (three relations) and GROUP BY aggregation
SELECT
    c.Location,
    j.JobTitle,
    COUNT(j.JobID) as position_count,
    MIN(j.Salary) as min_salary,
    MAX(j.Salary) as max_salary,
    AVG(j.Salary) as avg_salary,
    AVG(r.Rating) as avg_rating,
    COUNT(r.ReviewID) as review_count
FROM Job j
JOIN Company c ON j.CompanyName = c.CompanyName
JOIN Review r ON j.JobID = r.JobID
WHERE j.Rating > 2.8
GROUP BY c.Location, j.JobTitle
HAVING AVG(r.Rating) >= 3.0
ORDER BY avg_salary DESC limit 15;
```

```
    -> JOIN Company c ON j.CompanyName = c.CompanyName
    -> JOIN Review r ON j.JobID = r.JobID
    -> WHERE j.Rating > 2.8
    -> GROUP BY c.Location, j.JobTitle
    -> HAVING AVG(r.Rating) >= 3.0
    -> ORDER BY avg_salary DESC limit 15;
+---------------------------+-----------------------------------------------------------------------------+----------------+------------+------------+------------+--------------------+-----------
--+
| Location                  | JobTitle                                                                    | position_count | min_salary | max_salary | avg_salary | avg_rating         | review_coun
t |
+---------------------------+-----------------------------------------------------------------------------+----------------+------------+------------+------------+--------------------+-----------
--+
| Seattle, WA               | Staff Site Reliability Engineer - Enterprise                                |              1 |     180000 |     180000 |     180000 |                4.2 |
1 |
| Centennial, CO            | Sr Prinicpal Software Engineer                                              |              1 |     178005 |     178005 |     178005 |                3.7 |
1 |
| San Jose, CA              | Sr. Software and Controls Engineer                                          |              2 |     165000 |     165000 |     165000 |               3.45 |
2 |
| Seattle, WA               | Frontend Staff Software Engineer - Jobseeker Profile                        |              2 |     164000 |     164000 |     164000 |                4.1 |
2 |
| Torrance, CA              | SOLUTIONS ARCHITECT, SITE PERFORMANCE                                       |              1 |     160000 |     160000 |     160000 |                4.6 |
1 |
| Remote                    | Senior Software Engineer (FT)                                               |              1 |     145000 |     145000 |     145000 |                4.1 |
1 |
| San Francisco Bay Area, CA | c# Engineer                                                                |              1 |     145000 |     145000 |     145000 |                  3 |
1 |
| Dallas, TX                | Senior Software Engineer - Cards                                            |              1 |     140000 |     140000 |     140000 |                4.7 |
1 |
| Aberdeen Proving Ground, MD | Software Engineer                                                         |              2 |     140000 |     140000 |     140000 | 3.9000000000000004 |
2 |
| Dallas, TX                | Staff 2 Linux Software Engineer (Architect) - Opportunity for Working Remotely Dallas, TX |              1 |     139000 |     139000 |     139000 |                4.3 |
1 |
| Illinois                  | Staff Software Engineer - "Enterprise SaaS built on Rails"                  |              1 |     137000 |     137000 |     137000 |                3.1 |
1 |
| Seattle, WA               | Solutions Engineer, Software                                                |              8 |     128200 |     153800 |   136850.5 | 3.6999999999999997 |
8 |
| Remote                    | Lead Front End Engineer                                                     |              2 |     135000 |     135000 |     135000 |               3.45 |
2 |
| Seattle, WA               | Senior Software Engineer - Enterprise Platform                              |              2 |     132000 |     132000 |     132000 |               3.35 |
2 |
| Seattle, WA               | Software Engineering Manager - Candidate Management                         |              2 |     132000 |     132000 |     132000 |                  3 |
2 |
+---------------------------+-----------------------------------------------------------------------------+----------------+------------+------------+------------+--------------------+-----------
--+
15 rows in set (0.02 sec)
```

-- Query 3: Job Upload Analysis with Admin Status

```
-- Purpose: Tracks job upload patterns and admin approval statistics
-- Uses: Multiple joins, GROUP BY aggregation, subquery
SELECT
    u.UserID,
    u.UserName,
    COUNT(uh.UploadID) as total_uploads,
    SUM(CASE WHEN j.ApprovalStatus = TRUE THEN 1 ELSE 0 END) as approved_uploads,
    (SELECT COUNT(DISTINCT JobTitle)
     FROM Job j2
     JOIN UploadedHistory uh2 ON j2.JobID = uh2.JobID
     WHERE uh2.UserID = u.UserID) as unique_job_types
FROM User u
JOIN UploadedHistory uh ON u.UserID = uh.UserID
JOIN Job j ON uh.JobID = j.JobID
WHERE u.Age < 45 and j.Sponsored = False
GROUP BY u.UserID, u.UserName
HAVING total_uploads >= 2
ORDER BY approved_uploads DESC limit 15;
```

```
mysql> SELECT
    ->     u.UserID,
    ->     u.UserName,
    ->     COUNT(uh.UploadID) as total_uploads,
    ->     SUM(CASE WHEN j.ApprovalStatus = TRUE THEN 1 ELSE 0 END) as approved_uploads,
    ->     (SELECT COUNT(DISTINCT JobTitle)
    ->      FROM Job j2
    ->      JOIN UploadedHistory uh2 ON j2.JobID = uh2.JobID
    ->      WHERE uh2.UserID = u.UserID) as unique_job_types
    -> FROM User u
    -> JOIN UploadedHistory uh ON u.UserID = uh.UserID
    -> JOIN Job j ON uh.JobID = j.JobID
    -> WHERE u.Age <45 and j.Sponsored = False
    -> GROUP BY u.UserID, u.UserName
    -> HAVING total_uploads >= 2
    -> ORDER BY approved_uploads DESC limit 15;
+--------------------------------------+------------------+---------------+------------------+------------------+
| UserID                               | UserName         | total_uploads | approved_uploads | unique_job_types |
+--------------------------------------+------------------+---------------+------------------+------------------+
| 014376c5-83b5-4fc3-9ef8-45d21df9c1d8 | christopher87    |             2 |                0 |                2 |
| 01fb8040-4e79-48df-b718-e127105f96b9 | gary20           |             4 |                0 |                4 |
| 032caa92-80b2-459e-bd54-d531693026b9 | ricejoshua       |             2 |                0 |                2 |
| 0344e378-b56a-4239-950c-185c6746ab3c | whiterandy       |             2 |                0 |                2 |
| 03920ef1-0223-4486-abef-66f087bef53f | andersonjoshua   |             4 |                0 |                4 |
| 05586b0a-ebb8-4d18-b3bd-8465348d69ae | sarahconley      |             2 |                0 |                2 |
| 06422e77-927d-4472-aa7f-736e98a2daf9 | rodriguezmiranda |             2 |                0 |                2 |
| 0879583c-867e-4a3b-bea9-7e0b58388940 | vincent84        |             2 |                0 |                2 |
| 094050c6-9d72-4386-9264-158f87ce855e | mitchelljames    |             2 |                0 |                2 |
| 0a6655f6-6783-49d9-9918-14ea5f7fe7a2 | fperez           |             2 |                0 |                2 |
| 0c7e976c-6fdc-4b8e-8268-ef461b8660dd | cblackwell       |             2 |                0 |                2 |
| 0cc4348a-3cdc-45bf-9968-6d3182a6616f | johnsoncorey     |             6 |                0 |                6 |
| 0d46bccc-89bf-4fb4-833a-7898cbd5b28d | mullinsdouglas   |             3 |                0 |                3 |
| 0ded2634-3292-4abd-9401-19e1c737a0a5 | zrose            |             3 |                0 |                3 |
| 0df6114d-b748-4eec-885f-79c3dbfd3355 | melvinhill       |             3 |                0 |                3 |
+--------------------------------------+------------------+---------------+------------------+------------------+
15 rows in set (0.02 sec)
```

-- Query 4: Simple Job Search with Regular and Sponsored Jobs

-- Purpose: Search both regular and
sponsored jobs

-- Uses: JOIN and SET operator (UNION)

```
SELECT j.JobTitle, j.CompanyName, j.Salary, c.Location
FROM Job j
JOIN Company c ON j.CompanyName = c.CompanyName
WHERE j.Sponsored = FALSE AND j.Salary >= 100000
UNION
SELECT j.JobTitle, j.CompanyName, j.Salary, c.Location
FROM Job j
JOIN Company c ON j.CompanyName = c.CompanyName
WHERE j.Sponsored = TRUE AND j.Salary >= 80000
ORDER BY Salary DESC limit 15;
```

```
mysql> SELECT j.JobTitle, j.CompanyName, j.Salary, c.Location
    -> FROM Job j
    -> JOIN Company c ON j.CompanyName = c.CompanyName
    -> WHERE j.Sponsored = FALSE AND j.Salary >= 100000
    -> UNION
    -> SELECT j.JobTitle, j.CompanyName, j.Salary, c.Location
    -> FROM Job j
    -> JOIN Company c ON j.CompanyName = c.CompanyName
    -> WHERE j.Sponsored = TRUE AND j.Salary >= 80000
    -> ORDER BY Salary DESC limit 15;
+--------------------------------------------------------------------------+----------------------+--------+--------------------+
| JobTitle                                                                 | CompanyName          | Salary | Location           |
+--------------------------------------------------------------------------+----------------------+--------+--------------------+
| Engineer Software/Principal Engineer Software                            | Northrop Grumman     | 338720 | San Antonio, TX    |
| C# Senior Software Developer (Contractor) - Front Office Trading /Risk   | DataAxxis            | 328500 | New York, NY       |
| Senior Engineer II - Nordstrom Fraud Technology                          | Nordstrom            | 289200 | Denver, CO         |
| Senior Software Engineer - HR Tech                                       | Nordstrom            | 289200 | Denver, CO         |
| Sr 2 Software Engineer - Selling and Pricing                             | Nordstrom            | 289200 | Denver, CO         |
| iOS / Android Developer                                                  | Taoti                | 250000 | Washington, DC     |
| Principal Software Engineer                                              | Indeed               | 207000 | Seattle, WA        |
| Director of SW Engineering for Speech Startup                            | Hudson Paradise, LLC | 200000 | San Jose, CA       |
| Principal ML/Backend Software Engineering Manager (10+ years exp. req.)  | Audere               | 200000 | Seattle, WA        |
| Senior Machine Learning Engineer                                         | Dishcare             | 200000 | Palo Alto, CA      |
| Embedded Android Engineer                                                | Leia Inc             | 200000 | Menlo Park, CA     |
| DevOps SME / Architect                                                   | Nueducation Inc      | 200000 | Washington, DC     |
| Senior DevOps Engineer                                                   | RedPoint Global      | 200000 | Remote             |
| VP of Site Reliability Engineering, Sling TV                             | DISH                 | 192000 | American Fork, UT  |
| Senior Software Engineer                                                 | tastytrade           | 190000 | Chicago, IL        |
+--------------------------------------------------------------------------+----------------------+--------+--------------------+
15 rows in set (0.03 sec)
```

Part 2:

Query 1 Index:

0. Default cost: 463663.07

```
| -> Table scan on <temporary>  (actual time=5426.318..5426.369 rows=176 loops=1)
    -> Aggregate using temporary table  (actual time=5426.315..5426.315 rows=176 loops=1)
        -> Nested loop inner join  (cost=463663.07 rows=114877) (actual time=934.282..5368.386 rows=70460 loops=1)
            -> Filter: (j.Salary > (select #2))  (cost=346035.23 rows=1148771) (actual time=924.624..3075.171 rows=2713998 loops=1)
                -> Inner hash join (no condition)  (cost=346035.23 rows=1148771) (actual time=153.022..1192.948 rows=10827173 loops=1)
                    -> Filter: (j.CompanyName is not null)  (cost=1.20 rows=2751) (actual time=2.819..19.062 rows=8641 loops=1)
                        -> Table scan on j  (cost=1.20 rows=8253) (actual time=2.816..17.362 rows=8641 loops=1)
                    -> Hash
                        -> Table scan on u  (cost=143.30 rows=1253) (actual time=97.864..150.020 rows=1253 loops=1)
                -> Select #2 (subquery in condition; run only once)
                    -> Aggregate: avg(Job.Salary)  (cost=2069.60 rows=1) (actual time=771.202..771.202 rows=1 loops=1)
                        -> Table scan on Job  (cost=1244.30 rows=8253) (actual time=0.019..769.942 rows=8641 loops=1)
            -> Filter: (u.Location = c.Location)  (cost=0.00 rows=0.1) (actual time=0.001..0.001 rows=0 loops=2713998)
                -> Single-row index lookup on c using PRIMARY (CompanyName=j.CompanyName)  (cost=0.00 rows=1) (actual time=0.000..0.000 rows=1 loops=2713998)
 |
```

1. Index using only Company.Location cost: 6682

```
| -> Table scan on <temporary>  (actual time=1190.581..1190.624 rows=176 loops=1)
    -> Aggregate using temporary table  (actual time=1190.578..1190.578 rows=176 loops=1)
        -> Nested loop inner join  (cost=6682.00 rows=3945) (actual time=90.862..1144.224 rows=70460 loops=1)
            -> Nested loop inner join  (cost=2539.18 rows=5337) (actual time=0.051..106.105 rows=103175 loops=1)
                -> Filter: (u.Location is not null)  (cost=129.80 rows=1253) (actual time=0.028..1.656 rows=1253 loops=1)
                    -> Table scan on u  (cost=129.80 rows=1253) (actual time=0.027..1.339 rows=1253 loops=1)
                -> Filter: (u.Location = c.Location)  (cost=1.50 rows=4) (actual time=0.006..0.078 rows=82 loops=1253)
                    -> Covering index lookup on c using comp_loc (Location=u.Location)  (cost=1.50 rows=4) (actual time=0.006..0.054 rows=82 loops=1253)
            -> Filter: (j.Salary > (select #2))  (cost=0.55 rows=1) (actual time=0.009..0.010 rows=1 loops=103175)
                -> Index lookup on j using CompanyName (CompanyName=c.CompanyName)  (cost=0.55 rows=2) (actual time=0.008..0.009 rows=2 loops=103175)
                -> Select #2 (subquery in condition; run only once)
                    -> Aggregate: avg(Job.Salary)  (cost=1755.35 rows=1) (actual time=3.831..3.831 rows=1 loops=1)
                        -> Table scan on Job  (cost=930.05 rows=8253) (actual time=0.039..3.268 rows=8641 loops=1)
 |
```

2 Index using only User.Location cost: 4850.24

```
| -> Table scan on <temporary>  (actual time=116.748..116.789 rows=176 loops=1)
    -> Aggregate using temporary table  (actual time=116.746..116.746 rows=176 loops=1)
        -> Nested loop inner join  (cost=4850.24 rows=9118) (actual time=3.916..79.167 rows=70460 loops=1)
            -> Nested loop inner join  (cost=1342.58 rows=2751) (actual time=3.894..15.350 rows=2166 loops=1)
                -> Filter: ((j.Salary > (select #2)) and (j.CompanyName is not null))  (cost=379.82 rows=2751) (actual time=3.868..9.252 rows=2166 loops=1)
                    -> Table scan on j  (cost=379.82 rows=8253) (actual time=0.019..4.271 rows=8641 loops=1)
                    -> Select #2 (subquery in condition; run only once)
                        -> Aggregate: avg(Job.Salary)  (cost=1755.35 rows=1) (actual time=3.831..3.831 rows=1 loops=1)
                            -> Table scan on Job  (cost=930.05 rows=8253) (actual time=0.011..3.295 rows=8641 loops=1)
                -> Filter: (c.Location is not null)  (cost=0.25 rows=1) (actual time=0.003..0.003 rows=1 loops=2166)
                    -> Single-row index lookup on c using PRIMARY (CompanyName=j.CompanyName)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=2166)
            -> Filter: (u.Location = c.Location)  (cost=0.94 rows=3) (actual time=0.004..0.027 rows=33 loops=2166)
                -> Covering index lookup on u using user_loc (Location=c.Location)  (cost=0.94 rows=3) (actual time=0.004..0.019 rows=33 loops=2166)
 |
```

3 Index using Company.Location + User.Location cost: 6682

```
| -> Table scan on <temporary>  (actual time=923.254..923.290 rows=176 loops=1)
    -> Aggregate using temporary table  (actual time=923.251..923.251 rows=176 loops=1)
        -> Nested loop inner join  (cost=6682.00 rows=3945) (actual time=3.907..867.148 rows=70460 loops=1)
            -> Nested loop inner join  (cost=2539.18 rows=5337) (actual time=0.061..92.271 rows=103175 loops=1)
                -> Filter: (u.Location is not null)  (cost=129.80 rows=1253) (actual time=0.039..1.057 rows=1253 loops=1)
                    -> Covering index scan on u using user_loc  (cost=129.80 rows=1253) (actual time=0.038..0.854 rows=1253 loops=1)
                -> Filter: (u.Location = c.Location)  (cost=1.50 rows=4) (actual time=0.004..0.068 rows=82 loops=1253)
                    -> Covering index lookup on c using comp_loc (Location=u.Location)  (cost=1.50 rows=4) (actual time=0.004..0.044 rows=82 loops=1253)
            -> Filter: (j.Salary > (select #2))  (cost=0.55 rows=1) (actual time=0.007..0.007 rows=1 loops=103175)
                -> Index lookup on j using CompanyName (CompanyName=c.CompanyName)  (cost=0.55 rows=2) (actual time=0.005..0.007 rows=2 loops=103175)
                -> Select #2 (subquery in condition; run only once)
                    -> Aggregate: avg(Job.Salary)  (cost=1755.35 rows=1) (actual time=3.790..3.791 rows=1 loops=1)
                        -> Table scan on Job  (cost=930.05 rows=8253) (actual time=0.013..3.260 rows=8641 loops=1)
|
```

4 Index using only Job.Salary cost: 775628.71

```
| -> Table scan on <temporary>  (actual time=982.315..982.359 rows=176 loops=1)
    -> Aggregate using temporary table  (actual time=982.312..982.312 rows=176 loops=1)
        -> Nested loop inner join  (cost=775628.71 rows=254114) (actual time=1.450..930.316 rows=70460 loops=1)
            -> Filter: (u.Location = c.Location)  (cost=436745.96 rows=436545) (actual time=1.398..55.721 rows=103175 loops=1)
                -> Inner hash join (<hash>(u.Location)=<hash>(c.Location))  (cost=436745.96 rows=436545) (actual time=1.394..19.605 rows=103175 loops=1)
                    -> Table scan on c  (cost=0.08 rows=3484) (actual time=0.050..2.582 rows=3456 loops=1)
                    -> Hash
                        -> Table scan on u  (cost=129.80 rows=1253) (actual time=0.054..0.833 rows=1253 loops=1)
            -> Filter: (j.Salary > (select #2))  (cost=0.55 rows=1) (actual time=0.007..0.008 rows=1 loops=103175)
                -> Index lookup on j using CompanyName (CompanyName=c.CompanyName)  (cost=0.55 rows=2) (actual time=0.006..0.008 rows=2 loops=103175)
                -> Select #2 (subquery in condition; run only once)
                    -> Aggregate: avg(Job.Salary)  (cost=1755.35 rows=1) (actual time=4.735..4.736 rows=1 loops=1)
                        -> Covering index scan on Job using job_sal  (cost=930.05 rows=8253) (actual time=0.027..3.930 rows=8641 loops=1)
|
```

5 Index using Job.Salary + User.Location cost: 4495.09

```
| -> Table scan on <temporary>  (actual time=123.560..123.605 rows=176 loops=1)
    -> Aggregate using temporary table  (actual time=123.557..123.557 rows=176 loops=1)
        -> Nested loop inner join  (cost=4495.09 rows=7180) (actual time=0.063..82.819 rows=70460 loops=1)
            -> Nested loop inner join  (cost=1733.06 rows=2166) (actual time=0.041..12.701 rows=2166 loops=1)
                -> Filter: ((j.Salary > (select #2)) and (j.CompanyName is not null))  (cost=974.96 rows=2166) (actual time=0.028..7.081 rows=2166 loops=1)
                    -> Index range scan on j using job_sal over (26574.09663233422 < Salary)  (cost=974.96 rows=2166) (actual time=0.026..6.489 rows=2166 loops=1)
                    -> Select #2 (subquery in condition; run only once)
                        -> Aggregate: avg(Job.Salary)  (cost=1755.35 rows=1) (actual time=2.546..2.546 rows=1 loops=1)
                            -> Covering index scan on Job using job_sal  (cost=930.05 rows=8253) (actual time=0.026..2.021 rows=8641 loops=1)
                -> Filter: (c.Location is not null)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=2166)
                    -> Single-row index lookup on c using PRIMARY (CompanyName=j.CompanyName)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=2166)
            -> Filter: (u.Location = c.Location)  (cost=0.94 rows=3) (actual time=0.005..0.030 rows=33 loops=2166)
                -> Covering index lookup on u using user_loc (Location=c.Location)  (cost=0.94 rows=3) (actual time=0.005..0.021 rows=33 loops=2166)
|
```

Of all the five indexes we designed, the index using Job.Salary + User.Location presents the lowest cost at 4495.09. However, indexing using only User.Location has a cost of 4850.24 which only lacks behind a small margin, which demonstrates adding Job.Salary has a minimal effect on improving the overall cost of the index. Therefore, for situations where maximum cost efficiency is needed, Job.Salary + User.Location is a better option; however, if space is a concern, User.Location may be a better choice.

Query 2 Index:

0. Default cost: 890

```
| -> Sort: avg_salary DESC  (actual time=13.235..13.291 rows=420 loops=1)
    -> Filter: (avg(r.Rating) >= 3.0)  (actual time=12.841..13.090 rows=420 loops=1)
        -> Table scan on <temporary>  (actual time=12.833..12.997 rows=797 loops=1)
            -> Aggregate using temporary table  (actual time=12.829..12.829 rows=797 loops=1)
                -> Nested loop inner join  (cost=890.10 rows=509) (actual time=0.070..10.773 rows=1015 loops=1)
                    -> Nested loop inner join  (cost=711.85 rows=509) (actual time=0.058..7.830 rows=1015 loops=1)
                        -> Filter: (r.JobID is not null)  (cost=177.05 rows=1528) (actual time=0.033..1.096 rows=1561 loops=1)
                            -> Table scan on r  (cost=177.05 rows=1528) (actual time=0.032..0.931 rows=1561 loops=1)
                        -> Filter: ((j.Rating > 2.8) and (j.CompanyName is not null))  (cost=0.25 rows=0.3) (actual time=0.004..0.004 rows=1 loops=1561)
                            -> Single-row index lookup on j using PRIMARY (JobID=r.JobID)  (cost=0.25 rows=1) (actual time=0.004..0.004 rows=1 loops=1561)
                    -> Single-row index lookup on c using PRIMARY (CompanyName=j.CompanyName)  (cost=0.25 rows=1) (actual time=0.003..0.003 rows=1 loops=1015)
|
```

1. Using only Company.Location cost: 890

```
| -> Sort: avg_salary DESC  (actual time=11.886..11.940 rows=420 loops=1)
    -> Filter: (avg(r.Rating) >= 3.0)  (actual time=11.525..11.758 rows=420 loops=1)
        -> Table scan on <temporary>  (actual time=11.518..11.677 rows=797 loops=1)
            -> Aggregate using temporary table  (actual time=11.516..11.516 rows=797 loops=1)
                -> Nested loop inner join  (cost=890.10 rows=509) (actual time=0.045..9.610 rows=1015 loops=1)
                    -> Nested loop inner join  (cost=711.85 rows=509) (actual time=0.038..7.103 rows=1015 loops=1)
                        -> Filter: (r.JobID is not null)  (cost=177.05 rows=1528) (actual time=0.021..1.016 rows=1561 loops=1)
                            -> Table scan on r  (cost=177.05 rows=1528) (actual time=0.021..0.788 rows=1561 loops=1)
                        -> Filter: ((j.Rating > 2.8) and (j.CompanyName is not null))  (cost=0.25 rows=0.3) (actual time=0.004..0.004 rows=1 loops=1561)
                            -> Single-row index lookup on j using PRIMARY (JobID=r.JobID)  (cost=0.25 rows=1) (actual time=0.003..0.003 rows=1 loops=1561)
                    -> Single-row index lookup on c using PRIMARY (CompanyName=j.CompanyName)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=1015)
|
```

2 Using only Review.Rating cost: 979

```
| -> Sort: avg_salary DESC  (actual time=11.377..11.433 rows=420 loops=1)
    -> Filter: (avg(r.Rating) >= 3.0)  (actual time=10.952..11.221 rows=420 loops=1)
        -> Table scan on <temporary>  (actual time=10.945..11.123 rows=797 loops=1)
            -> Aggregate using temporary table  (actual time=10.942..10.942 rows=797 loops=1)
                -> Nested loop inner join  (cost=979.22 rows=764) (actual time=0.043..9.102 rows=1015 loops=1)
                    -> Nested loop inner join  (cost=711.85 rows=764) (actual time=0.036..6.697 rows=1015 loops=1)
                        -> Filter: (r.JobID is not null)  (cost=177.05 rows=1528) (actual time=0.019..0.977 rows=1561 loops=1)
                            -> Table scan on r  (cost=177.05 rows=1528) (actual time=0.018..0.833 rows=1561 loops=1)
                        -> Filter: ((j.Rating > 2.8) and (j.CompanyName is not null))  (cost=0.25 rows=0.5) (actual time=0.003..0.004 rows=1 loops=1561)
                            -> Single-row index lookup on j using PRIMARY (JobID=r.JobID)  (cost=0.25 rows=1) (actual time=0.003..0.003 rows=1 loops=1561)
                    -> Single-row index lookup on c using PRIMARY (CompanyName=j.CompanyName)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=1015)
|
```

3 Using only Job.JobTitle cost: 890

```
| -> Sort: avg_salary DESC  (actual time=11.433..11.489 rows=420 loops=1)
    -> Filter: (avg(r.Rating) >= 3.0)  (actual time=11.080..11.303 rows=420 loops=1)
        -> Table scan on <temporary>  (actual time=11.073..11.230 rows=797 loops=1)
            -> Aggregate using temporary table  (actual time=11.071..11.071 rows=797 loops=1)
                -> Nested loop inner join  (cost=890.10 rows=509) (actual time=0.043..9.202 rows=1015 loops=1)
                    -> Nested loop inner join  (cost=711.85 rows=509) (actual time=0.035..6.719 rows=1015 loops=1)
                        -> Filter: (r.JobID is not null)  (cost=177.05 rows=1528) (actual time=0.019..0.930 rows=1561 loops=1)
                            -> Table scan on r  (cost=177.05 rows=1528) (actual time=0.018..0.773 rows=1561 loops=1)
                        -> Filter: ((j.Rating > 2.8) and (j.CompanyName is not null))  (cost=0.25 rows=0.3) (actual time=0.003..0.004 rows=1 loops=1561)
                            -> Single-row index lookup on j using PRIMARY (JobID=r.JobID)  (cost=0.25 rows=1) (actual time=0.003..0.003 rows=1 loops=1561)
                    -> Single-row index lookup on c using PRIMARY (CompanyName=j.CompanyName)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=1015)
 |
```

For query 2, we developed three indexing designs, but the cost remained unchanged, and for index using Review.Rating, the cost even increased. The possible reasons for this are:

- **Ineffectiveness for Joins and Aggregation:** The query's complexity with multiple joins and grouping limits the usefulness of single-column indexes, as large portions of the tables still need to be scanned.
- **Misalignment with Query Filters:** The indexes don't align well with the primary filters, joins, and grouping columns, reducing their efficiency for cost improvement.
- **Optimizer Behavior:** For the Review.Rating index, the optimizer may have chosen an inefficient access path, as the cost of maintaining and scanning this index outweighed its benefits for the query.
- **Low Selectivity on Rating:** If Rating has many similar values, the index adds little filtering benefit, making full or partial scans more costly.

Since indexing did not reduce cost, we chose not to use additional indexes.

Query 3 Index:

0. Default cost: 624.25

```
| -> Sort: approved_uploads DESC  (actual time=14.430..14.457 rows=189 loops=1)
    -> Filter: (total_uploads >= 2)  (actual time=14.269..14.365 rows=189 loops=1)
        -> Table scan on <temporary>  (actual time=14.266..14.334 rows=384 loops=1)
            -> Aggregate using temporary table  (actual time=14.264..14.264 rows=384 loops=1)
                -> Nested loop inner join  (cost=624.25 rows=71) (actual time=0.096..6.944 rows=677 loops=1)
                    -> Nested loop inner join  (cost=377.02 rows=706) (actual time=0.085..4.562 rows=677 loops=1)
                        -> Filter: (u.Age < 45)  (cost=129.80 rows=418) (actual time=0.045..0.669 rows=531 loops=1)
                            -> Table scan on u  (cost=129.80 rows=1253) (actual time=0.043..0.559 rows=1253 loops=1)
                        -> Filter: (uh.JobID is not null)  (cost=0.42 rows=2) (actual time=0.006..0.007 rows=1 loops=531)
                            -> Index lookup on uh using UserID (UserID=u.UserID)  (cost=0.42 rows=2) (actual time=0.006..0.007 rows=1 loops=531)
                    -> Filter: (j.Sponsored = false)  (cost=0.25 rows=0.1) (actual time=0.003..0.003 rows=1 loops=677)
                        -> Single-row index lookup on j using PRIMARY (JobID=uh.JobID)  (cost=0.25 rows=1) (actual time=0.003..0.003 rows=1 loops=677)
-> Select #2 (subquery in projection; dependent)
    -> Aggregate: count(distinct j2.JobTitle)  (cost=1.35 rows=1) (actual time=0.015..0.015 rows=1 loops=384)
        -> Nested loop inner join  (cost=1.18 rows=2) (actual time=0.010..0.014 rows=2 loops=384)
            -> Filter: (uh2.JobID is not null)  (cost=0.59 rows=2) (actual time=0.007..0.008 rows=2 loops=384)
                -> Index lookup on uh2 using UserID (UserID=u.UserID)  (cost=0.59 rows=2) (actual time=0.007..0.008 rows=2 loops=384)
            -> Single-row index lookup on j2 using PRIMARY (JobID=uh2.JobID)  (cost=0.31 rows=1) (actual time=0.003..0.003 rows=1 loops=677)
 |
```

1 Using only Job.JobTitle cost: 624.25

```
| -> Sort: approved_uploads DESC  (actual time=14.203..14.218 rows=189 loops=1)
    -> Filter: (total_uploads >= 2)  (actual time=14.029..14.139 rows=189 loops=1)
        -> Table scan on <temporary>  (actual time=14.026..14.107 rows=384 loops=1)
            -> Aggregate using temporary table  (actual time=14.025..14.025 rows=384 loops=1)
                -> Nested loop inner join  (cost=624.25 rows=71) (actual time=0.092..6.837 rows=677 loops=1)
                    -> Nested loop inner join  (cost=377.02 rows=706) (actual time=0.081..4.444 rows=677 loops=1)
                        -> Filter: (u.Age < 45)  (cost=129.80 rows=418) (actual time=0.033..0.644 rows=531 loops=1)
                            -> Table scan on u  (cost=129.80 rows=1253) (actual time=0.031..0.535 rows=1253 loops=1)
                        -> Filter: (uh.JobID is not null)  (cost=0.42 rows=2) (actual time=0.006..0.007 rows=1 loops=531)
                            -> Index lookup on uh using UserID (UserID=u.UserID)  (cost=0.42 rows=2) (actual time=0.006..0.007 rows=1 loops=531)
                    -> Filter: (j.Sponsored = false)  (cost=0.25 rows=0.1) (actual time=0.003..0.003 rows=1 loops=677)
                        -> Single-row index lookup on j using PRIMARY (JobID=uh.JobID)  (cost=0.25 rows=1) (actual time=0.003..0.003 rows=1 loops=677)
-> Select #2 (subquery in projection; dependent)
    -> Aggregate: count(distinct j2.JobTitle)  (cost=1.35 rows=1) (actual time=0.015..0.015 rows=1 loops=384)
        -> Nested loop inner join  (cost=1.18 rows=2) (actual time=0.010..0.013 rows=2 loops=384)
            -> Filter: (uh2.JobID is not null)  (cost=0.59 rows=2) (actual time=0.007..0.008 rows=2 loops=384)
                -> Index lookup on uh2 using UserID (UserID=u.UserID)  (cost=0.59 rows=2) (actual time=0.007..0.008 rows=2 loops=384)
            -> Single-row index lookup on j2 using PRIMARY (JobID=uh2.JobID)  (cost=0.31 rows=1) (actual time=0.003..0.003 rows=1 loops=677)
 |
```

2 Using only User.Age cost: 758.48

```
| -> Sort: approved_uploads DESC  (actual time=14.287..14.302 rows=189 loops=1)
    -> Filter: (total_uploads >= 2)  (actual time=14.132..14.228 rows=189 loops=1)
        -> Table scan on <temporary>  (actual time=14.129..14.197 rows=384 loops=1)
            -> Aggregate using temporary table  (actual time=14.127..14.127 rows=384 loops=1)
                -> Nested loop inner join  (cost=758.48 rows=90) (actual time=0.070..6.877 rows=677 loops=1)
                    -> Nested loop inner join  (cost=444.14 rows=898) (actual time=0.058..4.529 rows=677 loops=1)
                        -> Filter: (u.Age < 45)  (cost=129.80 rows=531) (actual time=0.031..0.639 rows=531 loops=1)
                            -> Table scan on u  (cost=129.80 rows=1253) (actual time=0.029..0.522 rows=1253 loops=1)
                        -> Filter: (uh.JobID is not null)  (cost=0.42 rows=2) (actual time=0.006..0.007 rows=1 loops=531)
                            -> Index lookup on uh using UserID (UserID=u.UserID)  (cost=0.42 rows=2) (actual time=0.006..0.007 rows=1 loops=531)
                    -> Filter: (j.Sponsored = false)  (cost=0.25 rows=0.1) (actual time=0.003..0.003 rows=1 loops=677)
                        -> Single-row index lookup on j using PRIMARY (JobID=uh.JobID)  (cost=0.25 rows=1) (actual time=0.003..0.003 rows=1 loops=677)
-> Select #2 (subquery in projection; dependent)
    -> Aggregate: count(distinct j2.JobTitle)  (cost=1.35 rows=1) (actual time=0.015..0.015 rows=1 loops=384)
        -> Nested loop inner join  (cost=1.18 rows=2) (actual time=0.010..0.013 rows=2 loops=384)
            -> Filter: (uh2.JobID is not null)  (cost=0.59 rows=2) (actual time=0.007..0.008 rows=2 loops=384)
                -> Index lookup on uh2 using UserID (UserID=u.UserID)  (cost=0.59 rows=2) (actual time=0.007..0.007 rows=2 loops=384)
            -> Single-row index lookup on j2 using PRIMARY (JobID=uh2.JobID)  (cost=0.31 rows=1) (actual time=0.003..0.003 rows=1 loops=677)
 |
```

3 Using only Job.Sponsored cost: 624.25

```
| -> Sort: approved_uploads DESC  (actual time=14.370..14.386 rows=189 loops=1)
    -> Filter: (total_uploads >= 2)  (actual time=14.209..14.304 rows=189 loops=1)
        -> Table scan on <temporary>  (actual time=14.206..14.273 rows=384 loops=1)
            -> Aggregate using temporary table  (actual time=14.203..14.203 rows=384 loops=1)
                -> Nested loop inner join  (cost=624.25 rows=353) (actual time=0.071..6.891 rows=677 loops=1)
                    -> Nested loop inner join  (cost=377.02 rows=706) (actual time=0.061..4.524 rows=677 loops=1)
                        -> Filter: (u.Age < 45)  (cost=129.80 rows=418) (actual time=0.036..0.656 rows=531 loops=1)
                            -> Table scan on u  (cost=129.80 rows=1253) (actual time=0.034..0.543 rows=1253 loops=1)
                        -> Filter: (uh.JobID is not null)  (cost=0.42 rows=2) (actual time=0.006..0.007 rows=1 loops=531)
                            -> Index lookup on uh using UserID (UserID=u.UserID)  (cost=0.42 rows=2) (actual time=0.006..0.007 rows=1 loops=531)
                    -> Filter: (j.Sponsored = false)  (cost=0.25 rows=0.5) (actual time=0.003..0.003 rows=1 loops=677)
                        -> Single-row index lookup on j using PRIMARY (JobID=uh.JobID)  (cost=0.25 rows=1) (actual time=0.003..0.003 rows=1 loops=677)
-> Select #2 (subquery in projection; dependent)
    -> Aggregate: count(distinct j2.JobTitle)  (cost=1.35 rows=1) (actual time=0.015..0.015 rows=1 loops=384)
        -> Nested loop inner join  (cost=1.18 rows=2) (actual time=0.010..0.014 rows=2 loops=384)
            -> Filter: (uh2.JobID is not null)  (cost=0.59 rows=2) (actual time=0.007..0.008 rows=2 loops=384)
                -> Index lookup on uh2 using UserID (UserID=u.UserID)  (cost=0.59 rows=2) (actual time=0.007..0.008 rows=2 loops=384)
            -> Single-row index lookup on j2 using PRIMARY (JobID=uh2.JobID)  (cost=0.31 rows=1) (actual time=0.003..0.003 rows=1 loops=677)
|
```

For query 3, we developed three indexing designs, but the cost remained unchanged, and for index using Job.Sponsored, the cost even increased. We summarize several reasons for this result:

- **Misalignment with Complex Joins and Aggregation**: The query includes multiple joins, aggregation, and a subquery, which limit the effectiveness of single-column indexes.
- **Limited Impact of Indexed Filters**: Indexing on Job.Sponsored and User.Age provides minimal benefit since these columns don't align with the primary grouping and filtering logic across multiple joins and subqueries.
- **Optimizer Choices**: The database optimizer may not use these indexes if it finds that full scans or other paths are more efficient for the complex operations.
- **Low Selectivity on User.Age**: Indexing User.Age is less efficient since many users may be under 45, leading to higher costs due to a larger dataset being accessed.

Since indexing did not reduce cost, we chose not to use additional indexes.

Query 4 Index:

0 Default cost: 2107.67

```
| -> Sort: Salary DESC  (cost=2672.89..2672.89 rows=550) (actual time=16.185..16.235 rows=540 loops=1)
    -> Table scan on <union temporary>  (cost=2107.68..2117.04 rows=550) (actual time=15.920..16.010 rows=540 loops=1)
        -> Union materialize with deduplication  (cost=2107.67..2107.67 rows=550) (actual time=15.916..15.916 rows=540 loops=1)
            -> Nested loop inner join  (cost=1026.33 rows=275) (actual time=0.070..8.828 rows=1249 loops=1)
                -> Filter: ((j.Sponsored = false) and (j.Salary >= 100000) and (j.CompanyName is not null))  (cost=930.05 rows=275) (actual time=0.050..5.829 rows=1249 loops=1)
                    -> Table scan on j  (cost=930.05 rows=8253) (actual time=0.038..4.832 rows=8641 loops=1)
                -> Single-row index lookup on c using PRIMARY (CompanyName=j.CompanyName)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=1249)
            -> Nested loop inner join  (cost=1026.33 rows=275) (actual time=4.723..4.723 rows=0 loops=1)
                -> Filter: ((j.Sponsored = true) and (j.Salary >= 80000) and (j.CompanyName is not null))  (cost=930.05 rows=275) (actual time=4.721..4.721 rows=0 loops=1)
                    -> Table scan on j  (cost=930.05 rows=8253) (actual time=0.024..4.142 rows=8641 loops=1)
                -> Single-row index lookup on c using PRIMARY (CompanyName=j.CompanyName)  (cost=0.25 rows=1) (never executed)
|
```

1 Using only Job.Sponsored cost: 963.13

```
| -> Sort: Salary DESC  (cost=2554.45..2554.45 rows=1376) (actual time=24.874..24.924 rows=540 loops=1)
    -> Table scan on <union temporary>  (cost=963.13..982.80 rows=1376) (actual time=24.637..24.724 rows=540 loops=1)
        -> Union materialize with deduplication  (cost=963.12..963.12 rows=1376) (actual time=24.634..24.634 rows=540 loops=1)
            -> Nested loop inner join  (cost=825.16 rows=1375) (actual time=0.096..22.246 rows=1249 loops=1)
                -> Filter: ((j.Salary >= 100000) and (j.CompanyName is not null))  (cost=343.84 rows=1375) (actual time=0.085..19.188 rows=1249 loops=1)
                    -> Index lookup on j using job_spon (Sponsored=false)  (cost=343.84 rows=4126) (actual time=0.062..18.436 rows=8641 loops=1)
                -> Single-row index lookup on c using PRIMARY (CompanyName=j.CompanyName)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=1249)
            -> Nested loop inner join  (cost=0.40 rows=0.3) (actual time=0.017..0.017 rows=0 loops=1)
                -> Filter: ((j.Salary >= 80000) and (j.CompanyName is not null))  (cost=0.28 rows=0.3) (actual time=0.016..0.016 rows=0 loops=1)
                    -> Index lookup on j using job_spon (Sponsored=true)  (cost=0.28 rows=1) (actual time=0.016..0.016 rows=0 loops=1)
                -> Single-row index lookup on c using PRIMARY (CompanyName=j.CompanyName)  (cost=0.55 rows=1) (never executed)
|
```

2 Using only Job.Salary cost: 1477.13

```
| -> Sort: Salary DESC  (cost=1758.38..1758.38 rows=298) (actual time=17.590..17.641 rows=540 loops=1)
    -> Table scan on <union temporary>  (cost=1477.13..1483.33 rows=298) (actual time=17.325..17.428 rows=540 loops=1)
        -> Union materialize with deduplication  (cost=1477.11..1477.11 rows=298) (actual time=17.320..17.320 rows=540 loops=1)
            -> Nested loop inner join  (cost=606.03 rows=125) (actual time=0.061..8.453 rows=1249 loops=1)
                -> Filter: ((j.Sponsored = false) and (j.CompanyName is not null))  (cost=562.31 rows=125) (actual time=0.047..5.726 rows=1249 loops=1)
                    -> Index range scan on j using job_sal over (100000 <= Salary), with index condition: (j.Salary >= 100000)  (cost=562.31 rows=1249) (actual time=0.033..5.493 rows=1249 loops=1)
                -> Single-row index lookup on c using PRIMARY (CompanyName=j.CompanyName)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=1249)
            -> Nested loop inner join  (cost=841.25 rows=173) (actual time=6.494..6.494 rows=0 loops=1)
                -> Filter: ((j.Sponsored = true) and (j.CompanyName is not null))  (cost=780.56 rows=173) (actual time=6.494..6.494 rows=0 loops=1)
                    -> Index range scan on j using job_sal over (80000 <= Salary), with index condition: (j.Salary >= 80000)  (cost=780.56 rows=1734) (actual time=0.025..6.377 rows=1734 loops=1)
                -> Single-row index lookup on c using PRIMARY (CompanyName=j.CompanyName)  (cost=0.25 rows=1) (never executed)
|
```

3 Using Job(Sponsored,Salary) cost: 1125.53

```
| -> Sort: Salary DESC  (cost=2554.61..2554.61 rows=1250) (actual time=10.599..10.649 rows=540 loops=1)
    -> Table scan on <union temporary>  (cost=1125.53..1143.64 rows=1250) (actual time=10.342..10.437 rows=540 loops=1)
        -> Union materialize with deduplication  (cost=1125.52..1125.52 rows=1250) (actual time=10.339..10.339 rows=540 loops=1)
            -> Nested loop inner join  (cost=999.46 rows=1249) (actual time=0.106..8.063 rows=1249 loops=1)
                -> Filter: (j.CompanyName is not null)  (cost=562.31 rows=1249) (actual time=0.088..5.590 rows=1249 loops=1)
                    -> Index range scan on j using job_sponsal over (Sponsored = 0 AND 100000 <= Salary), with index condition: ((j.Sponsored = false) and (j.Salary >= 100000))  (cost=562.31 rows=1249) (actual ti
me=0.086..5.472 rows=1249 loops=1)
                -> Single-row index lookup on c using PRIMARY (CompanyName=j.CompanyName)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=1249)
            -> Nested loop inner join  (cost=1.06 rows=1) (actual time=0.021..0.021 rows=0 loops=1)
                -> Filter: (j.CompanyName is not null)  (cost=0.71 rows=1) (actual time=0.020..0.020 rows=0 loops=1)
                    -> Index range scan on j using job_sponsal over (Sponsored = 1 AND 80000 <= Salary), with index condition: ((j.Sponsored = true) and (j.Salary >= 80000))  (cost=0.71 rows=1) (actual time=0.020
..0.020 rows=0 loops=1)
                -> Single-row index lookup on c using PRIMARY (CompanyName=j.CompanyName)  (cost=0.35 rows=1) (never executed)
|
```

Out of all the index designs, we observed that using only Job.Sponsored presents the lowest cost. The reason for this could be:

High Selectivity: It filters jobs into distinct sponsored and non-sponsored groups, reducing scan time.

Efficient UNION Processing: It enables efficient retrieval for each part of the UNION, focusing on the sponsored status directly.
Lower Overhead: Sponsored indexing reduces data scans more effectively than Salary alone, minimizing query costs.
We choose using only Job.Sponsored as our final index.

Fix suggestions in Stage 2:

In order to protect users' privacy and allow users to express their true thoughts about the jobs, all the reviews are now anonymous. Additionally, reviews are not an attribute of a job, because the same job can have multiple reviews. Now, the schema of Review is: **Review:** (ReviewID: VARCHAR(50)[PK], JobID: VARCHAR(255)[FK to Job.JobID], Content: VARCHAR(4096), Rating: Real**).** It is an entity.