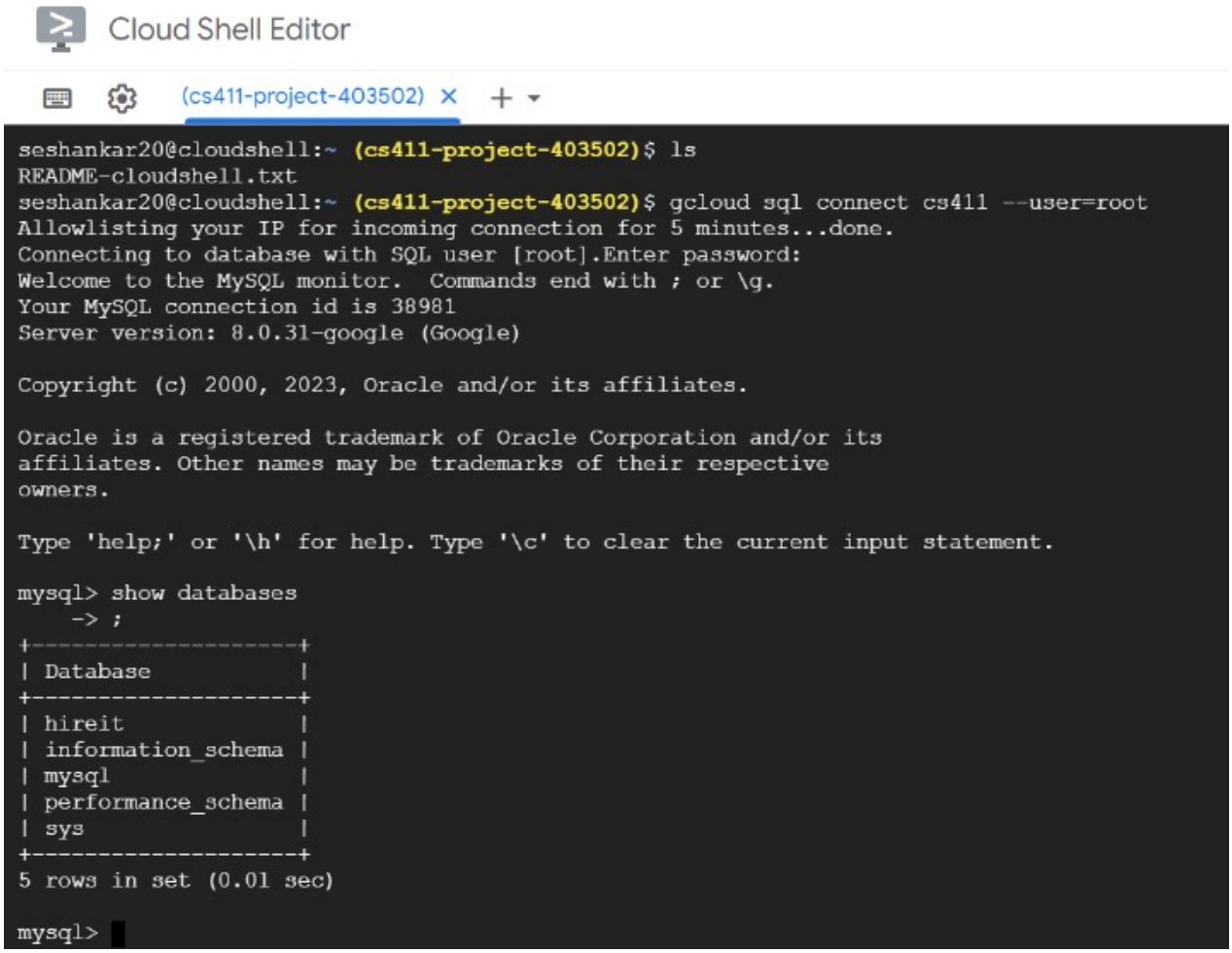
HireIt

Stage 3: Database Implementation and Indexing
Team No: 081
Team Name: ACID

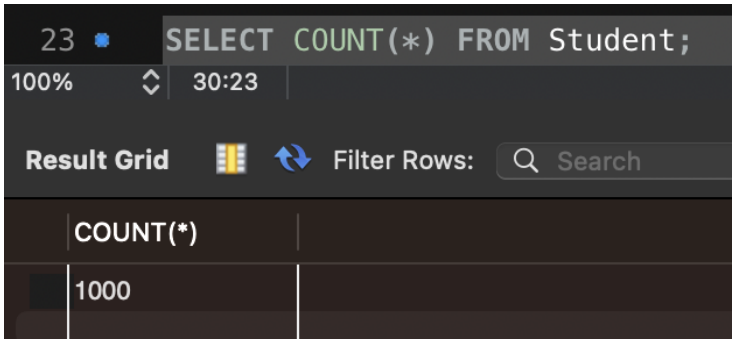
GCP Connection Details:



DDL Commands:

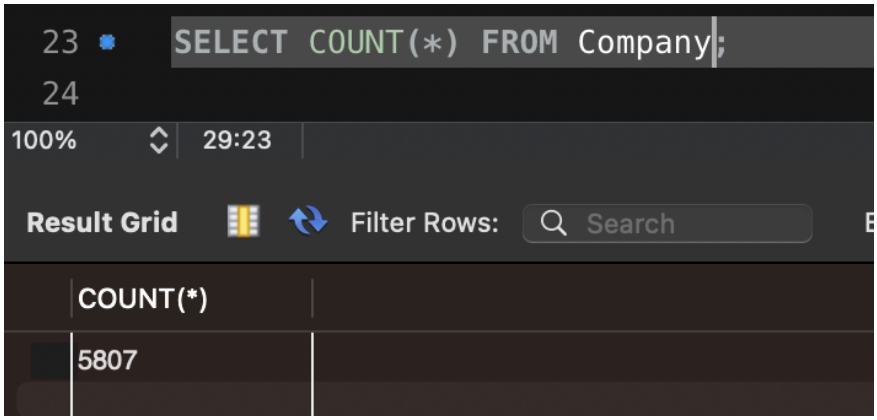
1. For Student table

```
CREATE TABLE IF NOT EXISTS Student (
    student_id INT NOT NULL,
    first_name VARCHAR(255) NOT NULL,
    last_name VARCHAR(255) NOT NULL,
    email VARCHAR(255) NOT NULL,
    gender VARCHAR(1) NOT NULL,
    dob DATE NOT NULL,
    age INT NOT NULL,
    university_name VARCHAR(255) NOT NULL,
    degree VARCHAR(255) NOT NULL,
    gpa DECIMAL(5, 2) NOT NULL,
    grad_date INT NOT NULL,
    pwd VARCHAR(20) NOT NULL,
    PRIMARY KEY (student_id)
);
```



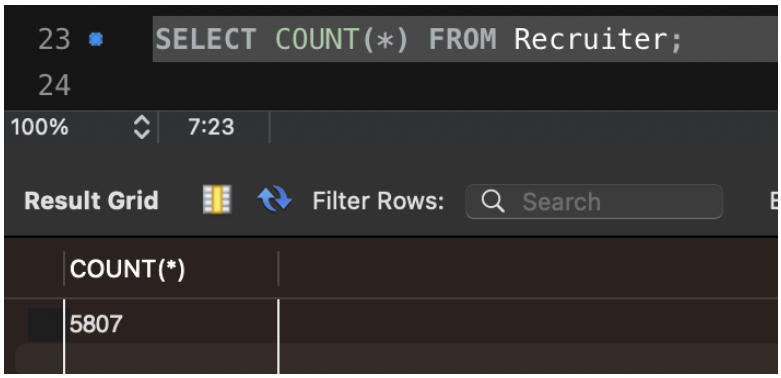
2. For Company table

```
CREATE TABLE IF NOT EXISTS Company (
    company_id INT NOT NULL,
    company_name VARCHAR(255) NOT NULL,
    hq_location VARCHAR(255) NOT NULL,
    sector VARCHAR(255) NOT NULL,
    PRIMARY KEY (company_id)
);
```



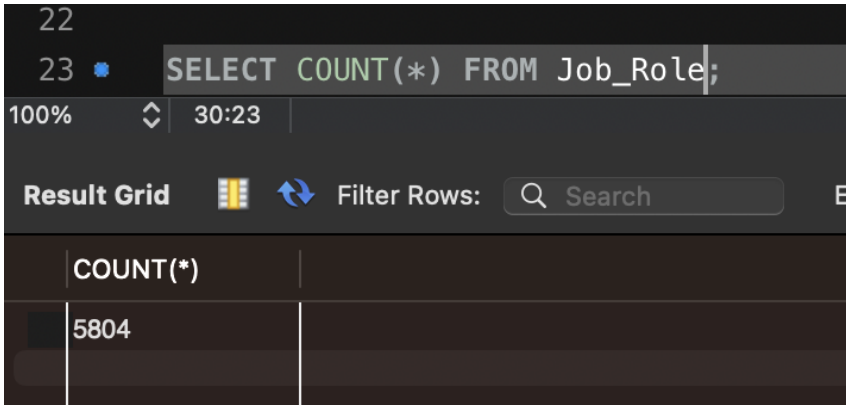
3. For Recruiter table

```
CREATE TABLE IF NOT EXISTS Recruiter(
   recruiter_id INT NOT NULL,
   first_name VARCHAR(255) NOT NULL,
   last_name VARCHAR(255) NOT NULL,
   email VARCHAR(255) NOT NULL,
   pwd VARCHAR(20) NOT NULL,
   company_id INT NOT NULL,
        FOREIGN KEY (company_id) REFERENCES Company(company_id) ON DELETE CASCADE ON UPDATE CASCADE,
   PRIMARY KEY (recruiter_id)
);
```



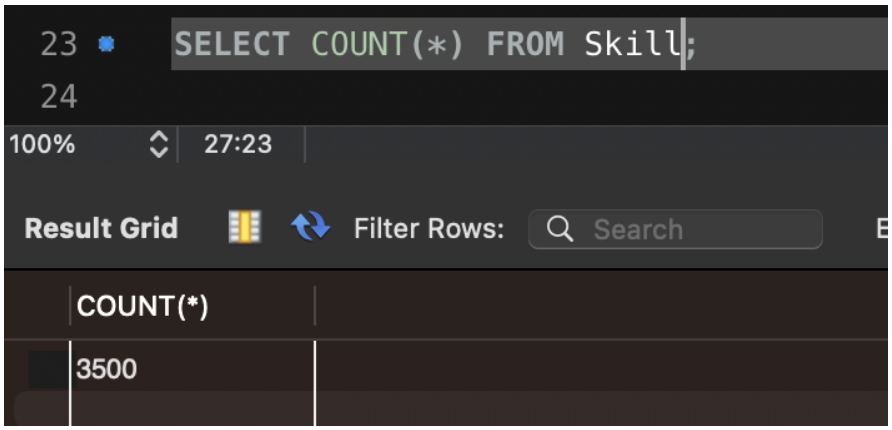
4. For Job_Role table

```
CREATE TABLE IF NOT EXISTS Job_Role(
   job_id INT NOT NULL,
   job_title VARCHAR(255) NOT NULL,
   salary INT NOT NULL,
   location VARCHAR(255) NOT NULL,
   job_type VARCHAR(255) NOT NULL,
   company_id INT NOT NULL,
        FOREIGN KEY (company_id) REFERENCES Company(company_id) ON DELETE
CASCADE ON UPDATE CASCADE,
   PRIMARY KEY (job_id)
);
```



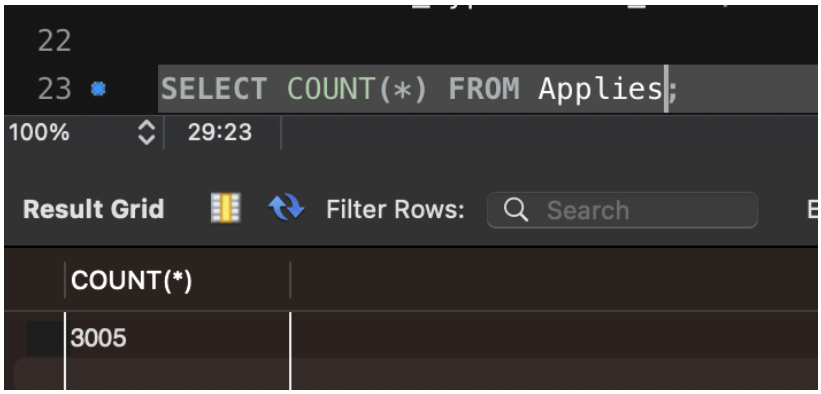
5. For Skill table

```
CREATE TABLE IF NOT EXISTS Skill(
   skill_id INT NOT NULL,
   skill_name VARCHAR(255) NOT NULL,
   PRIMARY KEY (skill_id)
);
```



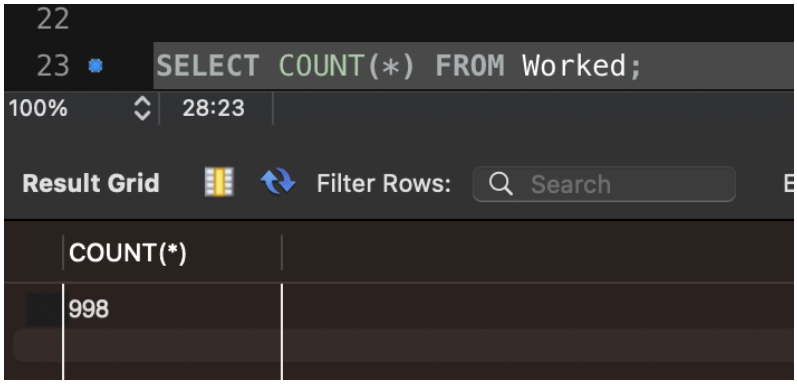
6. For Applies table

```
CREATE TABLE IF NOT EXISTS Applies (
   status VARCHAR(255) NOT NULL,
   student_id INT NOT NULL,
   job_id INT NOT NULL,
   FOREIGN KEY (student_id) REFERENCES Student(student_id) ON DELETE CASCADE ON UPDATE CASCADE,
      FOREIGN KEY (job_id) REFERENCES Job_Role(job_id) ON DELETE CASCADE ON UPDATE CASCADE,
   PRIMARY KEY (student_id, job_id)
);
```



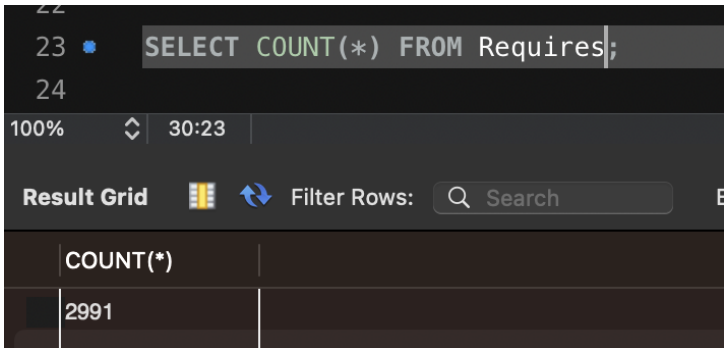
7. For Worked table

```
CREATE TABLE IF NOT EXISTS Worked (
   company_id INT NOT NULL,
   student_id INT NOT NULL,
   work_exp_id INT NOT NULL,
   role VARCHAR(255) NOT NULL,
   years_of_exp INT NOT NULL,
   job_type VARCHAR(255) NOT NULL,
      FOREIGN KEY (company_id) REFERENCES Company(company_id) ON DELETE CASCADE ON UPDATE CASCADE,
   FOREIGN KEY (student_id) REFERENCES Student(student_id) ON DELETE CASCADE ON UPDATE CASCADE,
   PRIMARY KEY (work_exp_id)
);
```



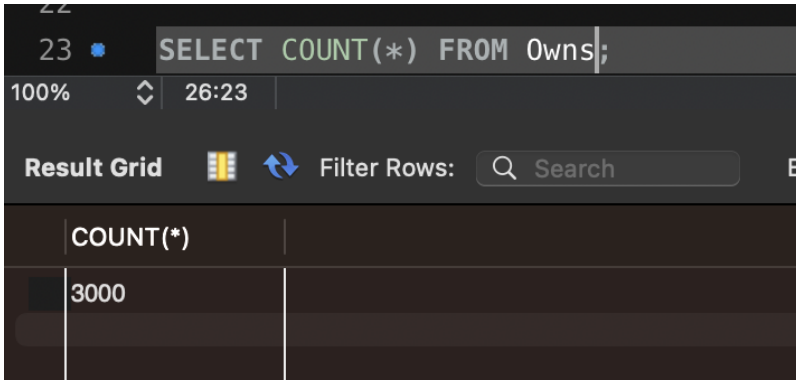
8. For Requires table

```
CREATE TABLE IF NOT EXISTS Requires(
   skill_id INT NOT NULL,
   job_id INT NOT NULL,
    FOREIGN KEY (skill_id) REFERENCES Skill(skill_id) ON DELETE CASCADE ON UPDATE CASCADE,
      FOREIGN KEY (job_id) REFERENCES Job_Role(job_id) ON DELETE CASCADE ON UPDATE CASCADE,
   PRIMARY KEY(skill_id, job_id)
);
```



9. For Owns table

```
CREATE TABLE IF NOT EXISTS Owns (
   student_id INT NOT NULL,
   skill_id INT NOT NULL,
   FOREIGN KEY (student_id) REFERENCES Student(student_id) ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY (skill_id) REFERENCES Skill(skill_id) ON DELETE CASCADE ON UPDATE CASCADE,
   PRIMARY KEY(student_id, skill_id)
);
```



Advanced Queries:

Query 1:
This query fetches the number of **Full Time** job roles posted by **Microsoft** with **salary>50000.** This query uses **Join** and **Aggregation.**

```
SELECT C.company_id, C.company_name, COUNT(*)
FROM Company C
JOIN Job_Role J ON J.company_id = C.company_id
WHERE C.company_id=9 and J.salary>50000 and J.job_type="Full Time"
GROUP BY C.company_id;
```

Output: This query returns only 1 row

company_id	company_name	COUNT(*)
9	microsoft	6

Query 2:
This query uses **JOIN** of multiple relations and Aggregation via **GROUP BY**

```
SELECT J.job_title, St.university_name, AVG(J.salary) as avg_salary
FROM Job_Role J
JOIN Applies A ON A.job_id = J.job_id
JOIN Student St ON A.student_id = St.student_id
```

```
WHERE A.status = 'Accepted'
GROUP BY J.job_title, St.university_name
ORDER BY avg_salary DESC
LIMIT 15;
```

Output (with limit 15):

Query 1

Result Grid | Filter Rows: Search | Export: | Fetch rows:

job_title	university_name	avg_salary
Finance and Administration Director	Hawaii Pacific University	99939.0000
IT Specialist	Mt San Antonio College	99917.0000
Administrative Assistant	Flair Beauty College	99892.0000
Chief of Party	Orleans Technical College	99825.0000
Monitoring and Evaluation Officer	Pima Medical Institution-San Antonio	99703.0000
Customs Clearance Officer	National American University-Albuquerque West	99646.0000
Manager	Tri-County Technical College	99583.0000
Logistics Driver	Saint Mary'S University Of Minnesota	99456.0000
HR Specialist	Sierra College Of Beauty	99275.0000
Software Developer	University Of Puerto Rico-Ponce	99258.0000
Administrative Officer	Community College Of Baltimore County	99231.0000
JSP/Java Developer for Lycos Billi...	Eastwick College-Ramsey	99214.0000
Health and Safety Manager	Seton Hall University	99187.0000
Instructor of Principles of Accounting	Altierus Career College-Tampa	99150.0000
Programmer/ Developer	Fulton-Montgomery Community College	99064.0000

Indexing Analysis:

Query 1:

```
SELECT C.company_id, C.company_name, COUNT(*)
FROM Company C
JOIN Job_Role J ON J.company_id = C.company_id
WHERE C.company_id=9 and J.salary>50000 and J.job_type="Full Time"
GROUP BY C.company_id;
```

Default index:

SHOW INDEX FROM Job_Role;

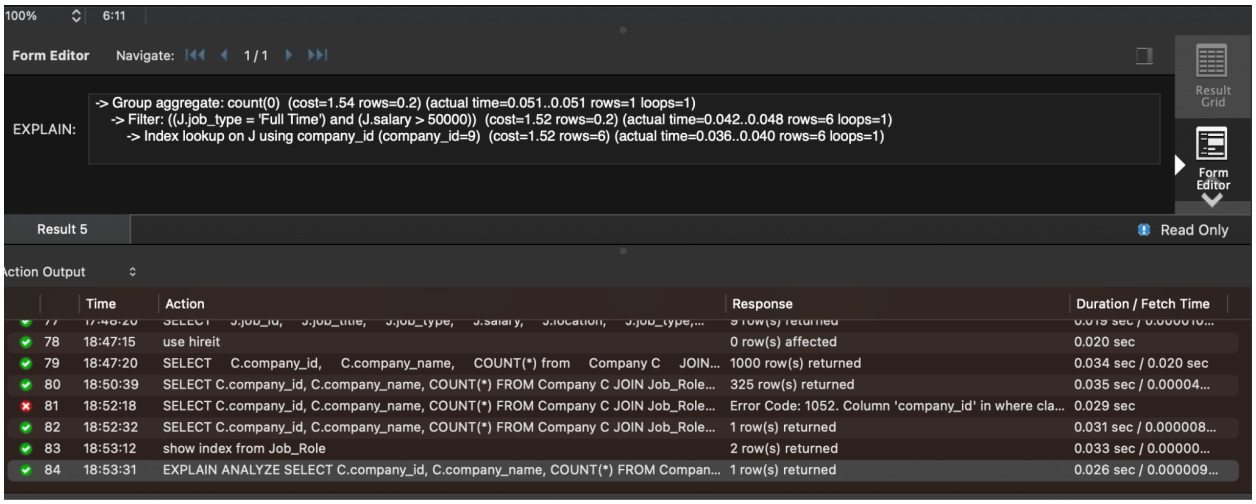
Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type	Comment	Index_comment	Visible	Expression
Job_Role	0	PRIMARY	1	job_id	A	6116	NULL	NULL		BTREE			YES	NULL
Job_Role	1	company_id	1	company_id	A	952	NULL	NULL		BTREE			YES	NULL

EXPLAIN ANALYZE execution time: 0.026 sec

18:53:31 EXPLAIN ANALYZE SELECT C.company_id, C.company_name, COUNT(*) FROM Company C JOIN Job_Role J ON J.company_id = C.company_id WHERE C.company_id=9 and J.salary>50000 and J.job_type="Full Time" GROUP BY C.company_id
1 row(s) returned 0.026 sec / 0.0000091 sec

EXPLAIN ANALYZE output:

'-> Group aggregate: count(0) (cost=1.54 rows=0.2) (actual time=0.051..0.051 rows=1 loops=1)\n -> Filter: ((J.job_type = \'Full Time\') and (J.salary > 50000)) (cost=1.52 rows=0.2) (actual time=0.042..0.048 rows=6 loops=1)\n -> Index lookup on J using company_id (company_id=9) (cost=1.52 rows=6) (actual time=0.036..0.040 rows=6 loops=1)\n'



Index 1: salary from Job_Role table

Table: Job_Role
Indexed column: salary
Index name: idx_salary

SHOW INDEX FROM Job_Role;

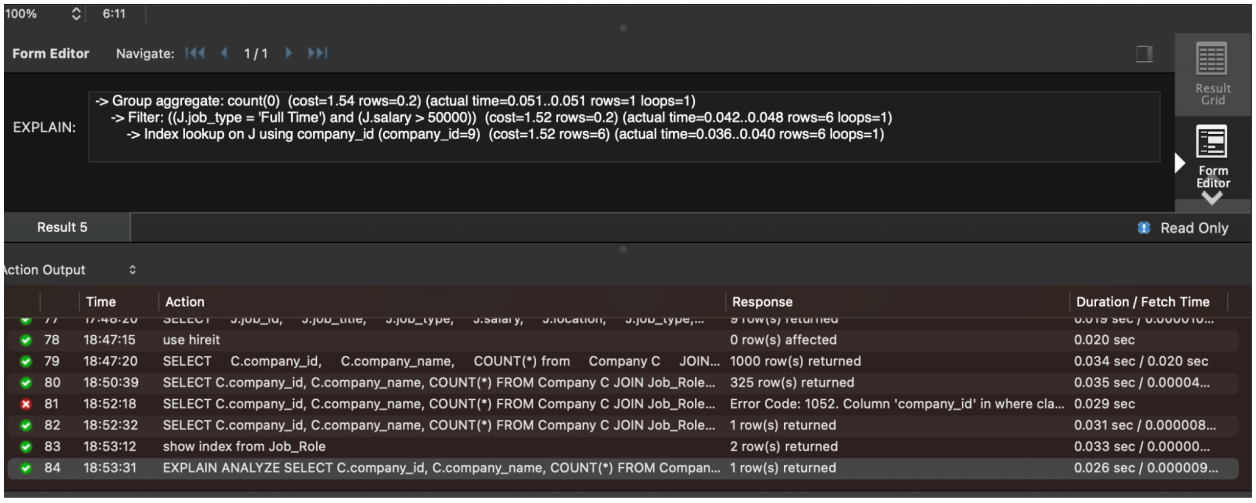
Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type	Comment	Index_comment	Visible	Expression
Job_Role	0	PRIMARY	1	job_id	A	6116	NULL	NULL		BTREE			YES	NULL
Job_Role	1	company_id	1	company_id	A	952	NULL	NULL		BTREE			YES	NULL
Job_Role	1	idx_salary	1	salary	A	5540	NULL	NULL		BTREE			YES	NULL

EXPLAIN ANALYZE execution time: 0.020 sec

18:55:30 EXPLAIN ANALYZE SELECT C.company_id, C.company_name, COUNT(*) FROM Company C JOIN Job_Role J ON J.company_id = C.company_id WHERE C.company_id=9 and J.salary>50000 and J.job_type="Full Time" GROUP BY C.company_id
1 row(s) returned **0.020 sec** / 0.0000038 sec

EXPLAIN ANALYZE output:

'-> Group aggregate: count(0) (cost=1.59 rows=0.5) (actual time=0.046..0.046 rows=1 loops=1)\n -> Filter: ((J.job_type = \'Full Time\') and (J.salary > 50000)) (cost=1.55 rows=0.5) (actual time=0.040..0.043 rows=6 loops=1)\n -> Index lookup on J using company_id (company_id=9) (cost=1.55 rows=6) (actual time=0.033..0.036 rows=6 loops=1)\n'



Index 2: job_type from Job_Role table

Table: Job_Role
Indexed column: job_type
Index name: idx_type

SHOW INDEX FROM Job_Role;

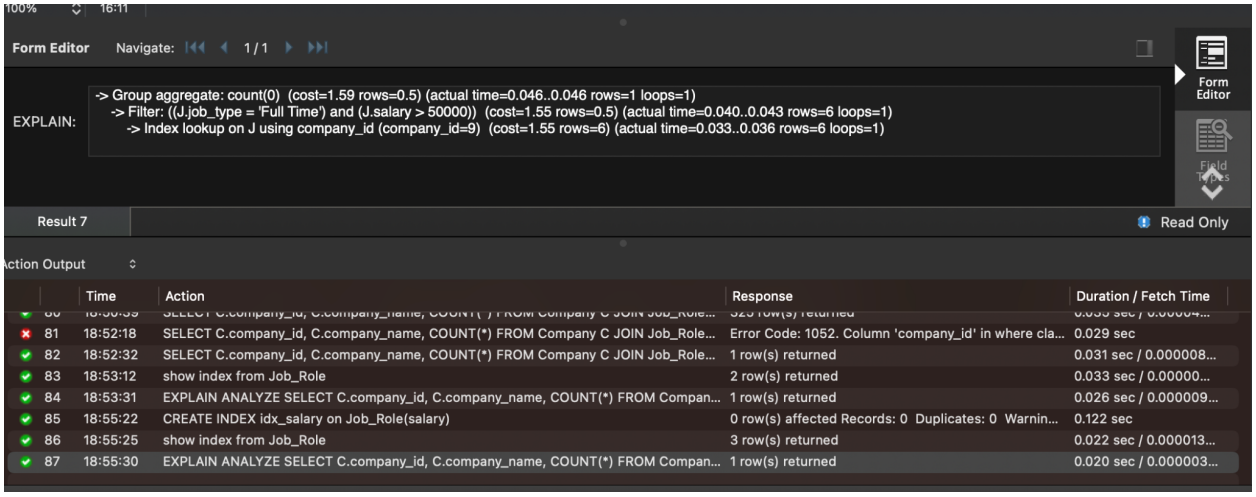
Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type	Comment	Index_comment	Visible	Expression
Job_Role	0	PRIMARY	1	job_id	A	6116	NULL	NULL		BTREE			YES	NULL
Job_Role	1	company_id	1	company_id	A	952	NULL	NULL		BTREE			YES	NULL
Job_Role	1	idx_type	1	job_type	A	3	NULL	NULL		BTREE			YES	NULL

EXPLAIN ANALYZE execution time: 0.024 sec

18:58:11 EXPLAIN ANALYZE SELECT C.company_id, C.company_name, COUNT(*) FROM Company C JOIN Job_Role J ON J.company_id = C.company_id WHERE C.company_id=9 and J.salary>50000 and J.job_type="Full Time" GROUP BY C.company_id
1 row(s) returned **0.024 sec** / 0.0000031 sec

EXPLAIN ANALYZE output:

'-> Group aggregate: count(0) (cost=1.62 rows=1) (actual time=0.040..0.040 rows=1 loops=1)\n -> Filter: ((J.job_type = \'Full Time\') and (J.salary > 50000)) (cost=1.56 rows=1) (actual time=0.033..0.037 rows=6 loops=1)\n -> Index lookup on J using company_id (company_id=9) (cost=1.56 rows=6) (actual time=0.030..0.033 rows=6 loops=1)\n'



Index 3: salary and job_type from Job_Role table

Table: Job_Role
Indexed columns: salary, job_type
Indices names: idx_salary, idx_type

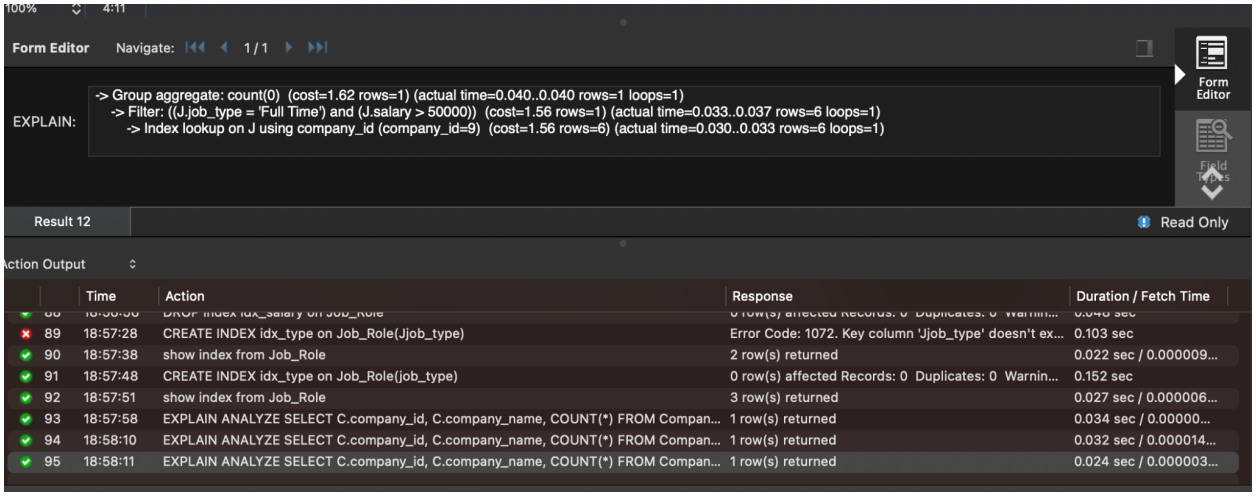
For the 3rd type of indexing, we have combined the first two indices idx_type and idx_salary.

SHOW INDEX FROM Job_Role;

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type	Comment	Index_comment	Visible	Expression
Job_Role	0	PRIMARY	1	job_id	A	6116	NULL	NULL		BTREE			YES	NULL
Job_Role	1	company_id	1	company_id	A	952	NULL	NULL		BTREE			YES	NULL
Job_Role	1	idx_type	1	job_type	A	3	NULL	NULL		BTREE			YES	NULL
Job_Role	1	idx_salary	1	salary	A	5540	NULL	NULL		BTREE			YES	NULL

EXPLAIN ANALYZE execution time: 0.019 sec
19:00:06 EXPLAIN ANALYZE SELECT C.company_id, C.company_name, COUNT(*) FROM Company C JOIN Job_Role J ON J.company_id = C.company_id WHERE C.company_id=9 and J.salary>50000 and J.job_type="Full Time" GROUP BY C.company_id
1 row(s) returned **0.019 sec** / 0.0000038 sec

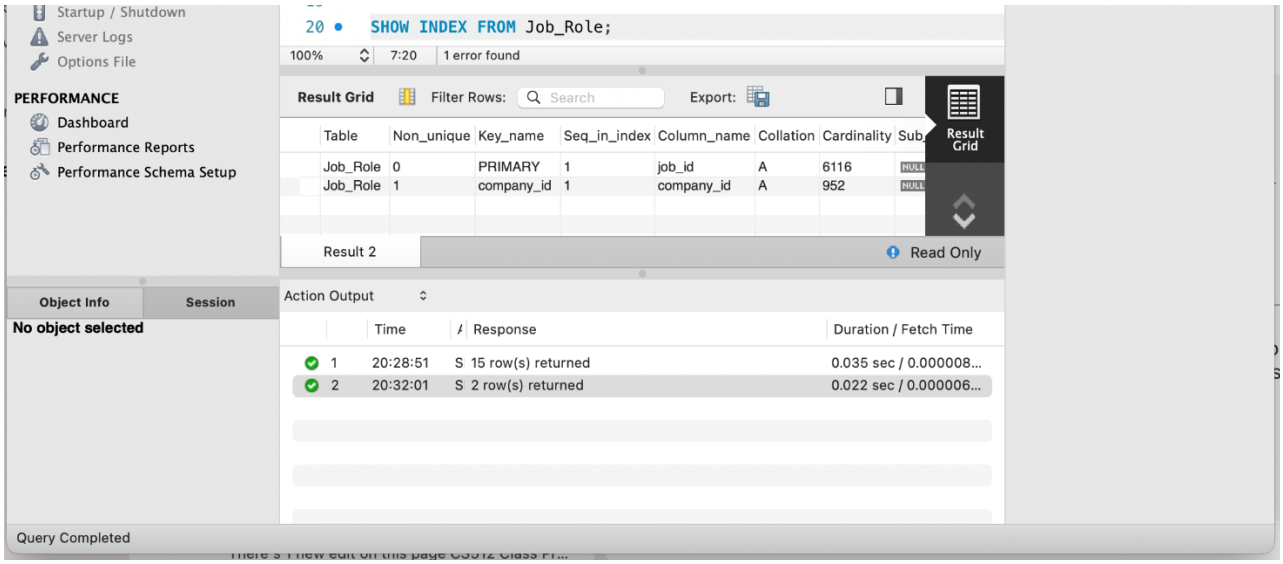
EXPLAIN ANALYZE output:
'-> Group aggregate: count(0) (cost=1.79 rows=1) (actual time=0.040..0.040 rows=1 loops=1)\n -> Filter: ((J.job_type = \'Full Time\') and (J.salary > 50000)) (cost=1.65 rows=1) (actual time=0.031..0.037 rows=6 loops=1)\n -> Index lookup on J using company_id (company_id=9) (cost=1.65 rows=6) (actual time=0.029..0.032 rows=6 loops=1)\n'



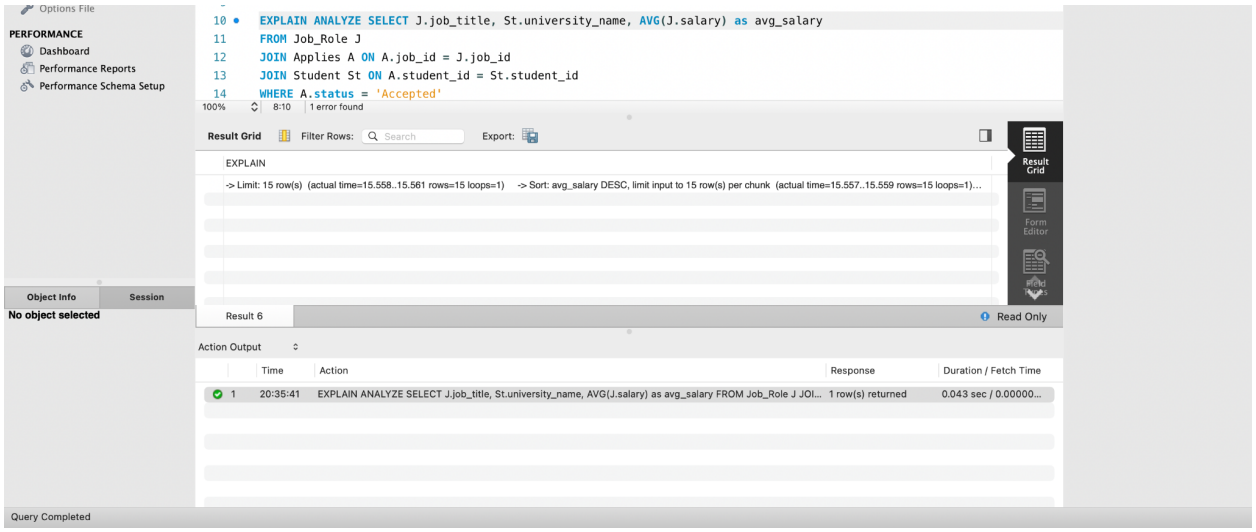
Query 2:

```
SELECT J.job_title, St.university_name, AVG(J.salary) as avg_salary
FROM Job_Role J
JOIN Applies A ON A.job_id = J.job_id
JOIN Student St ON A.student_id = St.student_id
WHERE A.status = 'Accepted'
GROUP BY J.job_title, St.university_name
ORDER BY avg_salary DESC
LIMIT 15;
```

Default index:
SHOW INDEX FROM Job_Role;



EXPLAIN ANALYZE execution time: 0.043 sec



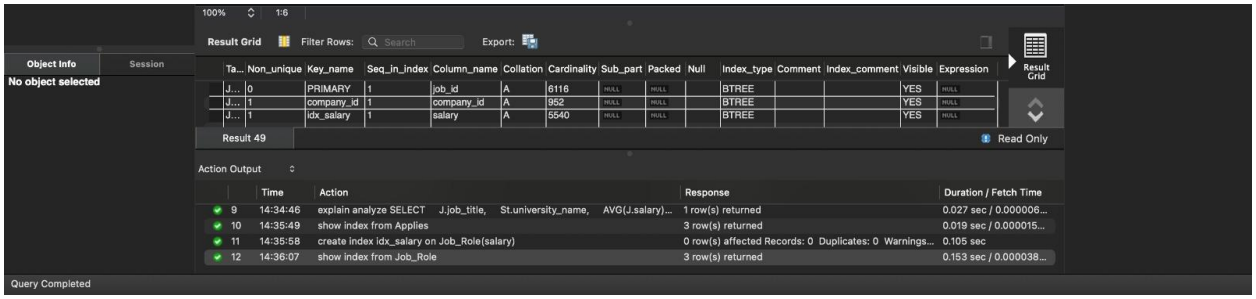
EXPLAIN ANALYZE output:

'-> Limit: 15 row(s) (actual time=15.558..15.561 rows=15 loops=1)\n -> Sort: avg_salary DESC, limit input to 15 row(s) per chunk (actual time=15.557..15.559 rows=15 loops=1)\n -> Table scan on <temporary> (actual time=14.683..14.887 rows=1029 loops=1)\n -> Aggregate using temporary table (actual time=14.677..14.677 rows=1029 loops=1)\n -> Nested loop inner join (cost=473.07 rows=277) (actual time=0.124..11.833 rows=1030 loops=1)\n -> Nested loop inner join (cost=376.09 rows=277) (actual time=0.114..9.478 rows=1030 loops=1)\n -> Filter: (A.`status` = \'Accepted\') (cost=279.10 rows=277) (actual time=0.096..2.512 rows=1030 loops=1)\n -> Table scan on A (cost=279.10 rows=2771) (actual time=0.089..1.735 rows=3005 loops=1)\n -> Single-row index lookup on St using PRIMARY (student_id=A.student_id) (cost=0.25 rows=1) (actual time=0.006..0.007 rows=1 loops=1030)\n -> Single-row index lookup on J using PRIMARY (job_id=A.job_id) (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=1030)\n'

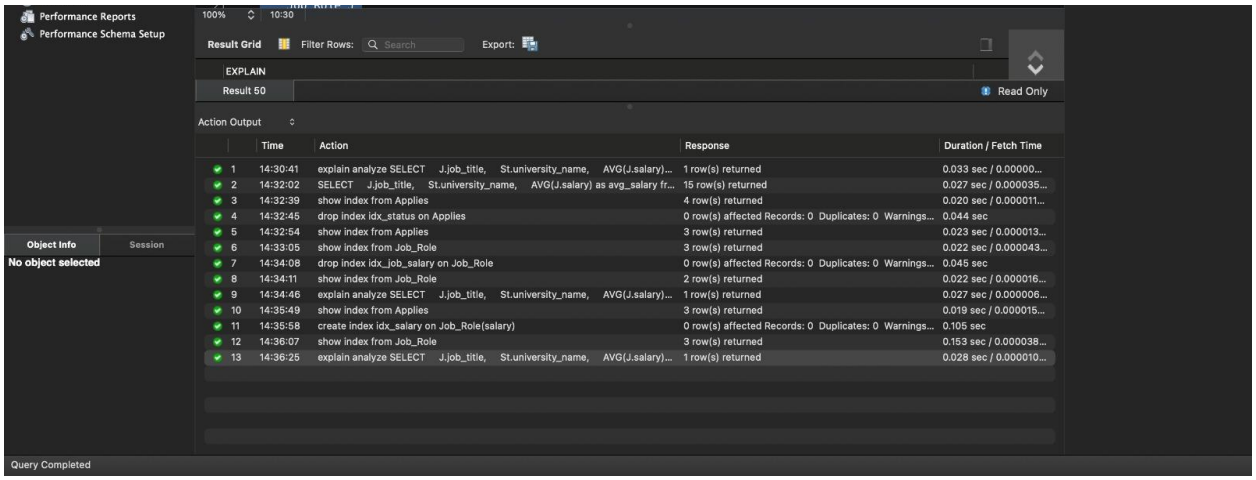
Index 1: salary from Job_Role table

Table: Job_Role
Indexed columns: salary
Indices names: idx_salary

SHOW INDEX FROM Job_Role;



EXPLAIN ANALYZE execution time: 0.028 sec



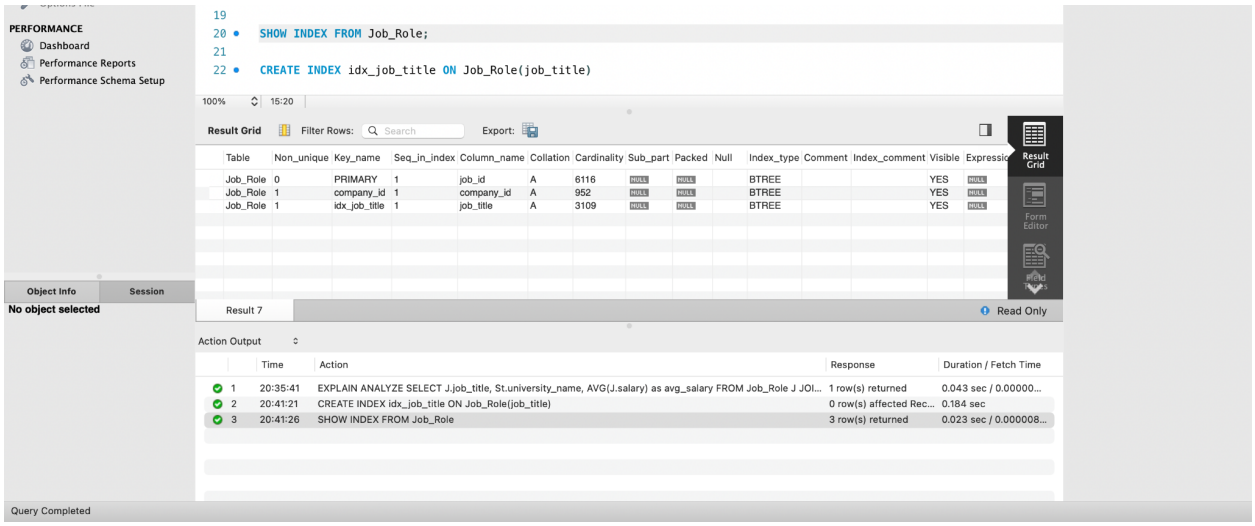
EXPLAIN ANALYZE output:

'-> Limit: 15 row(s) (actual time=6.427..6.429 rows=15 loops=1)\n -> Sort: avg_salary DESC, limit input to 15 row(s) per chunk (actual time=6.426..6.427 rows=15 loops=1)\n -> Table scan on <temporary> (actual time=5.923..6.083 rows=1029 loops=1)\n -> Aggregate using temporary table (actual time=5.920..5.920 rows=1029 loops=1)\n -> Nested loop inner join (cost=473.07 rows=277) (actual time=0.141..4.155 rows=1030 loops=1)\n -> Nested loop inner join (cost=376.09 rows=277) (actual time=0.120..2.656 rows=1030 loops=1)\n -> Filter: (A.`status` = \'Accepted\') (cost=279.10 rows=277) (actual time=0.099..1.402 rows=1030 loops=1)\n -> Table scan on A (cost=279.10 rows=2771) (actual time=0.092..0.948 rows=2997 loops=1)\n -> Single-row index lookup on St using PRIMARY (student_id=A.student_id) (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=1030)\n -> Single-row index lookup on J using PRIMARY (job_id=A.job_id) (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=1030)\n'

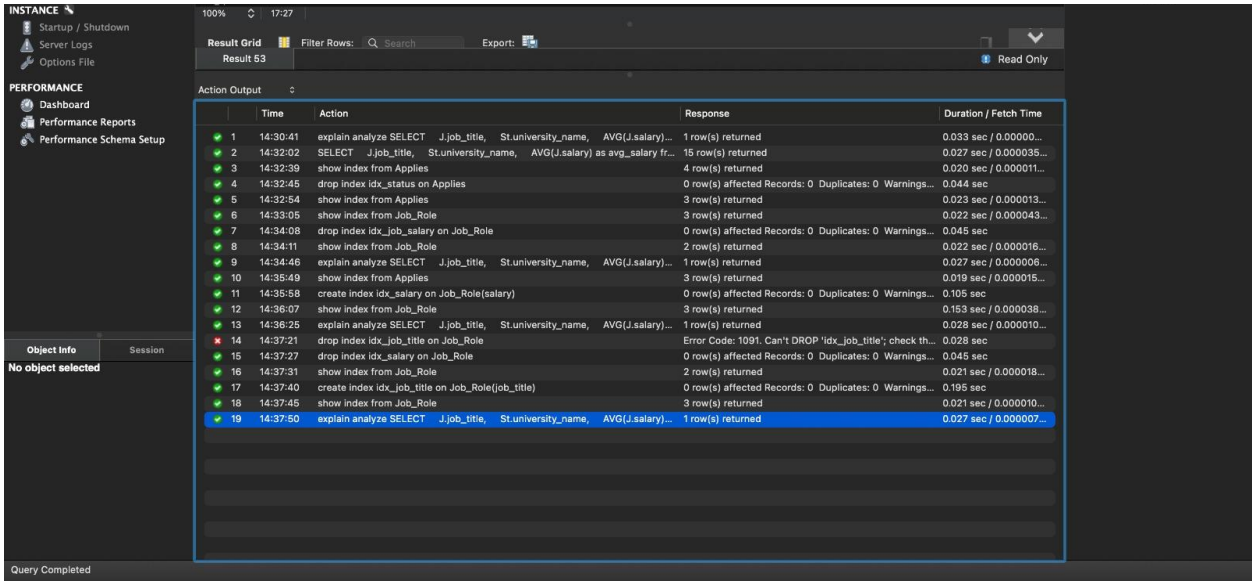
Index 2: job_title from Job_Role table

Table: Job_Role
Indexed columns: job_title

Indices names: idx_job_title
SHOW INDEX FROM Job_Role;



EXPLAIN ANALYZE execution time: 0.027 sec



EXPLAIN ANALYZE output:

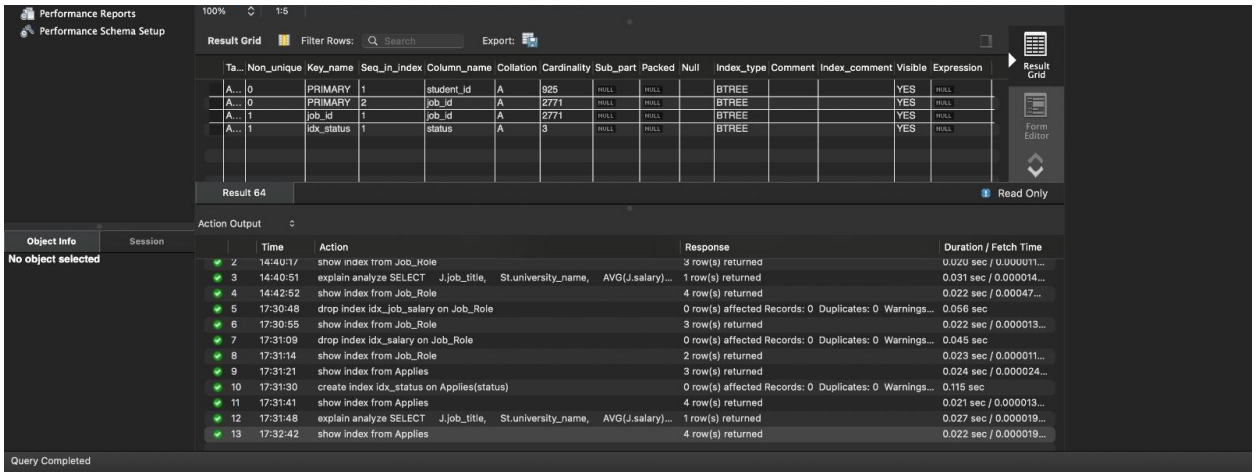
'-> Limit: 15 row(s) (actual time=7.536..7.539 rows=15 loops=1)\n -> Sort: avg_salary DESC, limit input to 15 row(s) per chunk (actual time=7.536..7.537 rows=15 loops=1)\n -> Table scan on <temporary> (actual time=7.012..7.168 rows=1029 loops=1)\n -> Aggregate using temporary table (actual time=7.009..7.009 rows=1029 loops=1)\n -> Nested loop inner join (cost=473.07 rows=277) (actual time=0.155..5.007 rows=1030 loops=1)\n -> Nested loop inner join (cost=376.09 rows=277) (actual time=0.144..3.192 rows=1030 loops=1)\n -> Filter: (A.`status` = \'Accepted\') (cost=279.10 rows=277) (actual time=0.117..1.588 rows=1030 loops=1)\n -> Table scan on A (cost=279.10 rows=2771) (actual time=0.107..1.038 rows=2997 loops=1)\n -> Single-row index

lookup on St using PRIMARY (student_id=A.student_id) (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=1030)\n -> Single-row index lookup on J using PRIMARY (job_id=A.job_id) (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=1030)\n'

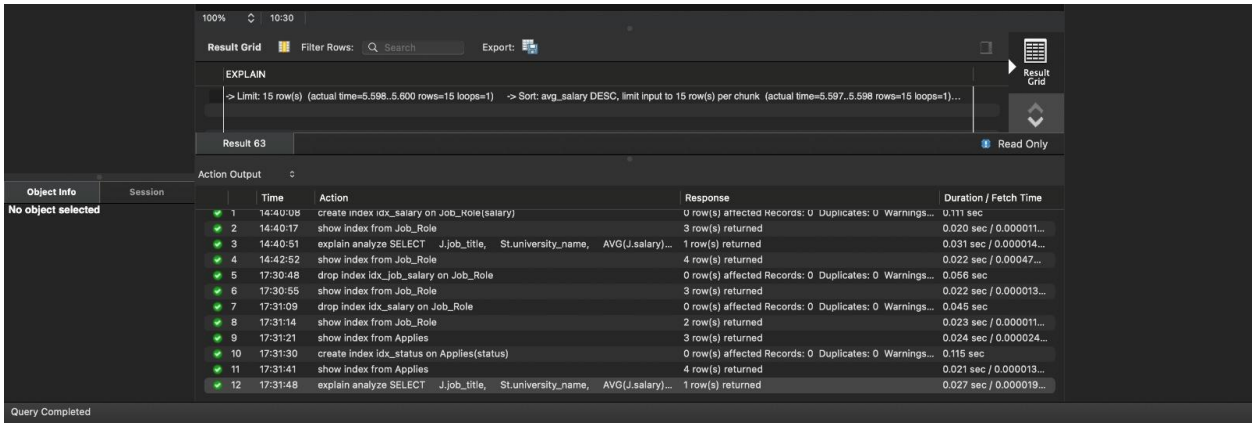
Index 3: status from Applies table

Table: Applies
Indexed columns: status
Indices names: idx_status

SHOW INDEX FROM Applies;



EXPLAIN ANALYZE execution time: 0.027 sec



EXPLAIN ANALYZE output:
'-> Limit: 15 row(s) (actual time=5.598..5.600 rows=15 loops=1)\n -> Sort: avg_salary DESC, limit input to 15 row(s) per chunk (actual time=5.597..5.598 rows=15 loops=1)\n -> Table scan on <temporary> (actual time=5.071..5.235 rows=1029 loops=1)\n -> Aggregate using temporary table (actual time=5.067..5.067 rows=1029 loops=1)\n -> Nested loop inner join (cost=934.18 rows=1030) (actual time=0.081..3.300 rows=1030 loops=1)\n

-> Nested loop inner join (cost=573.68 rows=1030) (actual time=0.073..1.777 rows=1030 loops=1)\n -> Covering index lookup on A using idx_status (status=\'Accepted\') (cost=213.18 rows=1030) (actual time=0.060..0.496 rows=1030 loops=1)\n -> Single-row index lookup on St using PRIMARY (student_id=A.student_id) (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=1030)\n -> Single-row index lookup on J using PRIMARY (job_id=A.job_id) (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=1030)\n'

Indexing Observation:

Query 1:

1. Index on column "salary":

The "salary" column is used in our queries' WHERE clauses.
The query performance with an index on "salary" improved to 0.020 seconds from the default 0.026 seconds. This improvement in performance could be due to the fact that data is efficiently filtered and retrieved using salary index.

2. Index on column "job_type":

The query performance with an index on "job_type" improved to 0.024 seconds, which is still better than the default. But, it is slightly slower than just indexing "salary." This could be due to the data distribution and query patterns; "salary" may have more distinct values, making it more efficient to filter on.

3. Index using both "salary" and "job_type":

The query performance further improved to 0.019 seconds, which is slightly better than indexing just "salary" alone. This suggests that for queries involving both "salary" and "job_type," the composite index is beneficial.This composite index helps the database to locate rows that match both conditions efficiently.

Index chosen:

Index using both "salary" and "job_type":
We chose to implement the composite indexing design using both salary and job_type as this method gave the best results.

Query 2:

1. Index on column "salary":

The query finds the average wage using the "salary" field. By indexing it, the database can easily find and compile wage numbers, which is essential for figuring out the average.
From the default 0.043 seconds, the query performance with an index on "salary" was increased to 0.028 seconds. This suggests that indexing the "salary" will accelerate the aggregate process and query execution.

2. Index on column "job_title":

The query's GROUP BY clause makes use of the "job_title" column. Data is effectively grouped by job title with the use of indexing. Compared to the default, the query speed improved to 0.027 seconds using an index on "job_title". More effective categorization is made possible by this index, which is important for figuring out the average pay for each job title.

3. Index on column "status":

The WHERE clause filters for "Accepted" applications using the "status" column. The performance of the query with an index on "status" was much better, reaching 0.0027 seconds. By indexing it, the speed increases the execution of queries by reducing the time required to choose the data.

Index chosen:
Index using “status”
We have chosen to implement indexing on “status” as there was a significant increase in the performance when compared to indexing “salary” and “job_title”