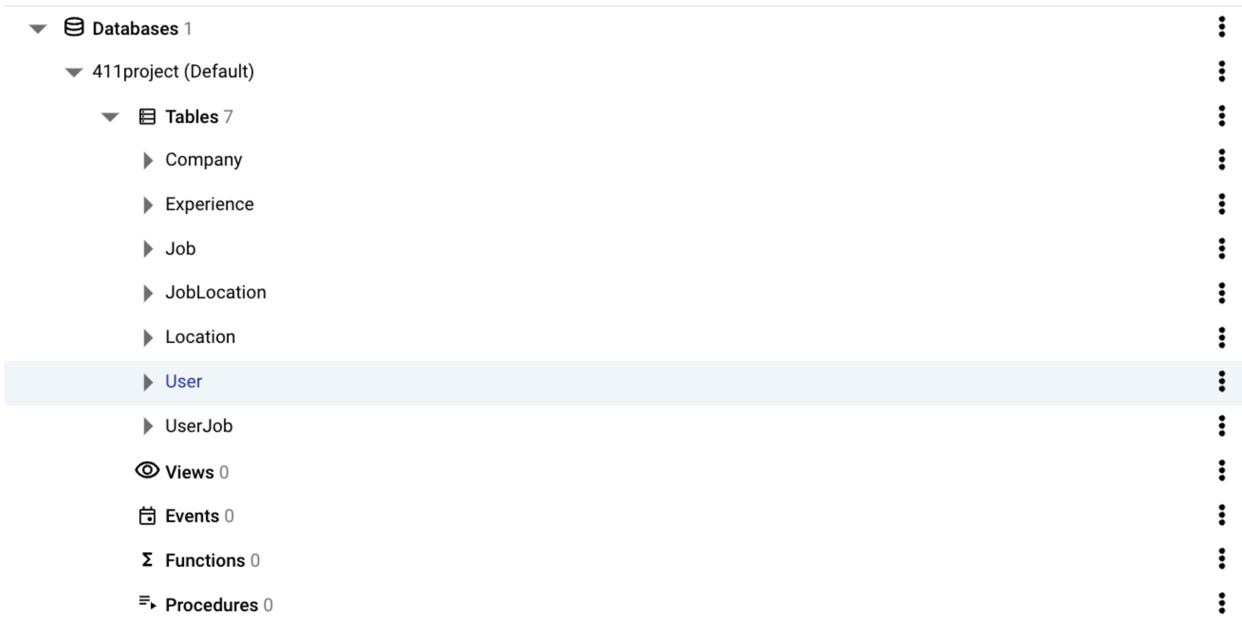
Stage 3: Database Implementation

DDL Commands

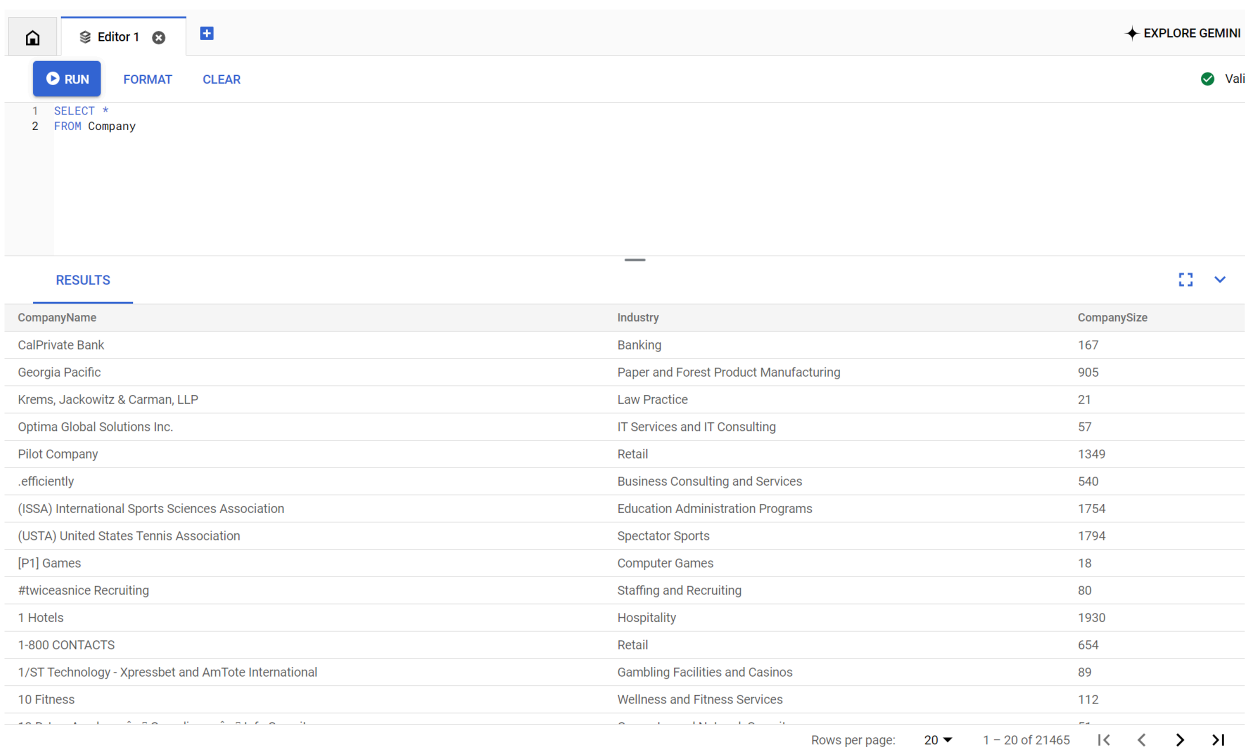
```sql
CREATE TABLE `411project`.`User` (
    UserId INT AUTO_INCREMENT PRIMARY KEY,
    Resume TEXT,
    Email VARCHAR(255) NOT NULL UNIQUE,
    Password VARCHAR(255) NOT NULL,
    FirstName VARCHAR(255),
    LastName VARCHAR(255)
);


CREATE TABLE `411project`.`Company` (
    CompanyName VARCHAR(255) PRIMARY KEY,
    Industry VARCHAR(255),
    CompanySize INT
);

CREATE TABLE `411project`.`Location` (
    LocationId INT AUTO_INCREMENT PRIMARY KEY,
    City VARCHAR(255),
    State VARCHAR(255),
    ZipCode VARCHAR(10),
    Country VARCHAR(255)
);

CREATE TABLE `411project`.`Experience` (
    ExperienceId INT AUTO_INCREMENT PRIMARY KEY,
    UserId INT NOT NULL,
    StartDate DATE,
    EndDate DATE,
    Achievement TEXT,
    Skills TEXT,
    FOREIGN KEY (UserId) REFERENCES `User`(UserId) ON DELETE CASCADE
```

```sql
);

CREATE TABLE `411project`.`Job` (
    JobId UNSIGNED INT AUTO_INCREMENT PRIMARY KEY,
    CompanyName VARCHAR(255) NOT NULL,
    JobRole VARCHAR(255),
    LocationId INT,
    Description TEXT,
    MinSalary DECIMAL(10, 2),
    MaxSalary DECIMAL(10, 2),
    Skills TEXT,
    FOREIGN KEY (CompanyName) REFERENCES `Company`(CompanyName),
    FOREIGN KEY (LocationId) REFERENCES `Location`(LocationId)
);

CREATE TABLE `411project`.JobLocation (
    JobId UNSIGNED INT UNSIGNED NOT NULL,
    LocationId INT NOT NULL,
    PRIMARY KEY (JobId, LocationId),
    FOREIGN KEY (JobId) REFERENCES Job(JobId),
    FOREIGN KEY (LocationId) REFERENCES Location(LocationId)
);

CREATE TABLE `411project`.`UserJob` (
    UserId INT NOT NULL,
    JobId UNSIGNED INT NOT NULL,
    ApplicationDate DATE,
    Status VARCHAR(50),
    PRIMARY KEY (UserId, JobId),
    FOREIGN KEY (UserId) REFERENCES `User`(UserId) ON DELETE CASCADE,
    FOREIGN KEY (JobId) REFERENCES `Job`(JobId) ON DELETE CASCADE
);
```

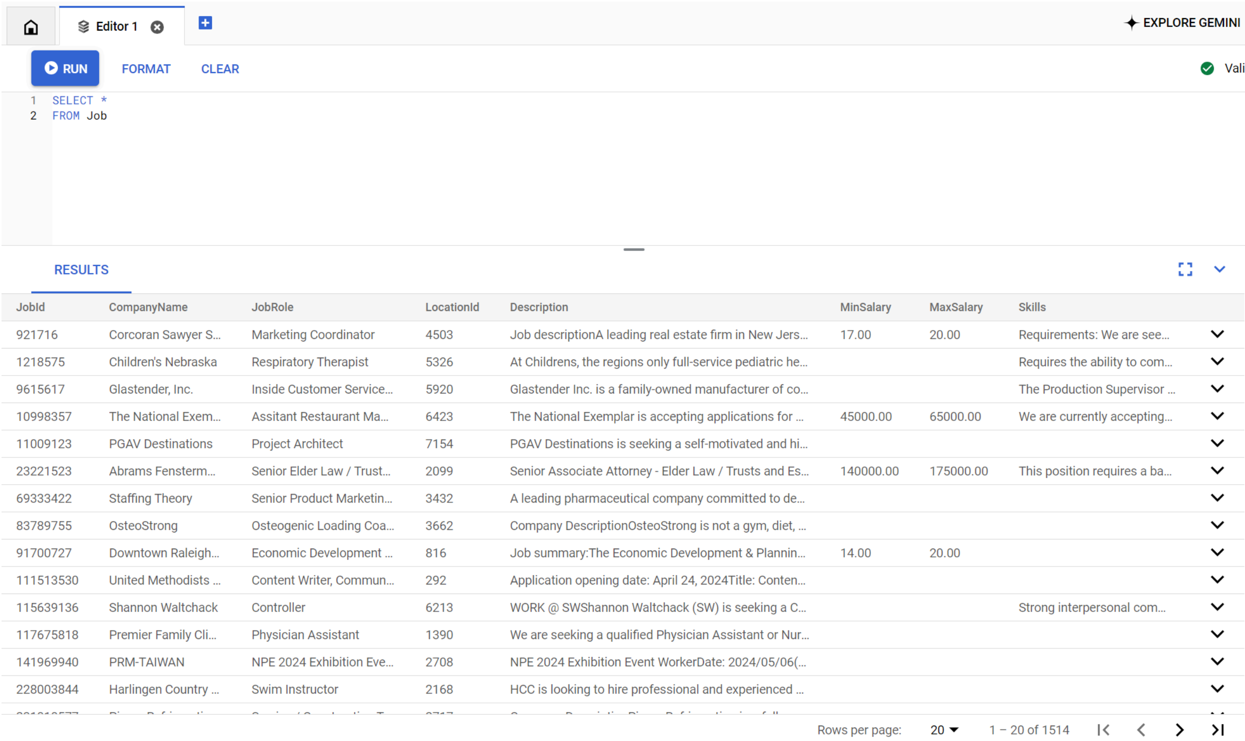
Screenshot of Database Implementation



Screenshot of Tables With Over 1000 Rows



The company table has 21465 rows.



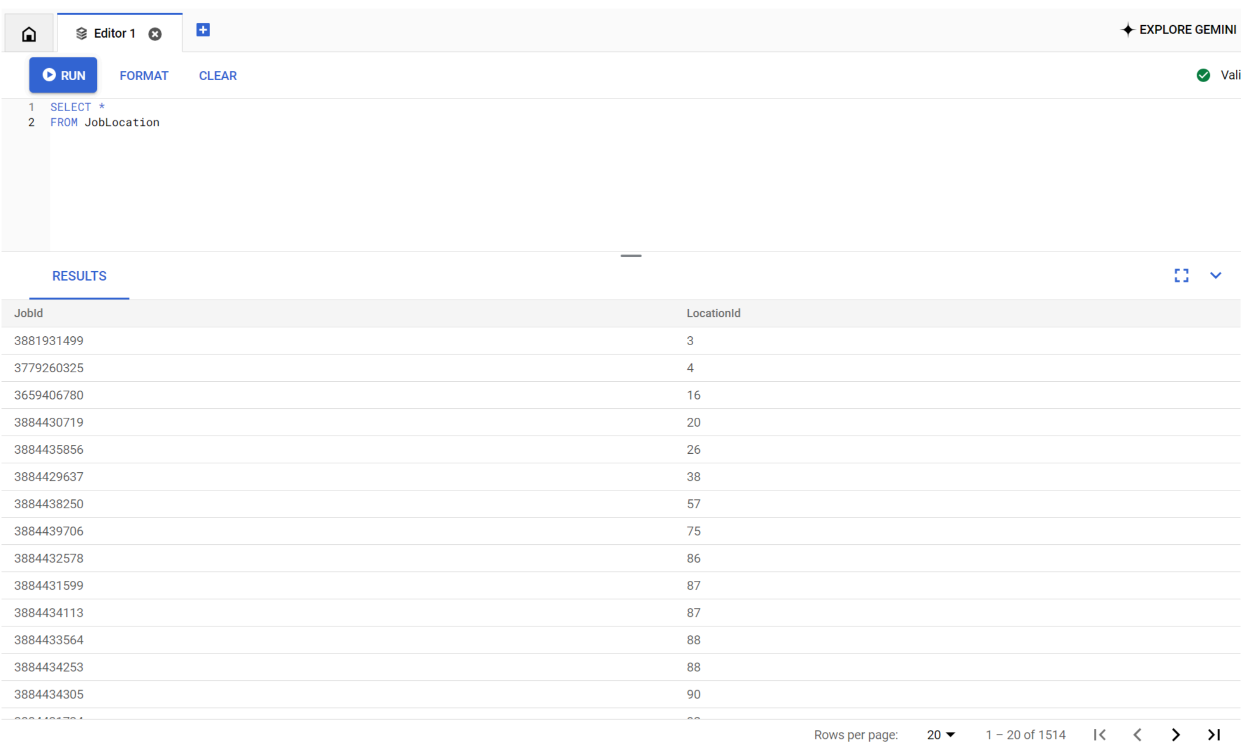
```
SELECT *
FROM Job
```

RESULTS

JobId	CompanyName	JobRole	LocationId	Description	MinSalary	MaxSalary	Skills
921716	Corcoran Sawyer S...	Marketing Coordinator	4503	Job descriptionA leading real estate firm in New Jers...	17.00	20.00	Requirements: We are see...
1218575	Children's Nebraska	Respiratory Therapist	5326	At Childrens, the regions only full-service pediatric he...			Requires the ability to com...
9615617	Glastender, Inc.	Inside Customer Service...	5920	Glastender Inc. is a family-owned manufacturer of co...			The Production Supervisor ...
10998357	The National Exem...	Assitant Restaurant Ma...	6423	The National Exemplar is accepting applications for ...	45000.00	65000.00	We are currently accepting...
11009123	PGAV Destinations	Project Architect	7154	PGAV Destinations is seeking a self-motivated and hi...			
23221523	Abrams Fensterm...	Senior Elder Law / Trust...	2099	Senior Associate Attorney - Elder Law / Trusts and Es...	140000.00	175000.00	This position requires a ba...
69333422	Staffing Theory	Senior Product Marketin...	3432	A leading pharmaceutical company committed to de...			
83789755	OsteoStrong	Osteogenic Loading Coa...	3662	Company DescriptionOsteoStrong is not a gym, diet, ...			
91700727	Downtown Raleigh...	Economic Development ...	816	Job summary:The Economic Development & Plannin...	14.00	20.00	
111513530	United Methodists ...	Content Writer, Commun...	292	Application opening date: April 24, 2024Title: Conten...			
115639136	Shannon Waltchack	Controller	6213	WORK @ SWShannon Waltchack (SW) is seeking a C...			Strong interpersonal com...
117675818	Premier Family Cli...	Physician Assistant	1390	We are seeking a qualified Physician Assistant or Nur...			
141969940	PRM-TAIWAN	NPE 2024 Exhibition Eve...	2708	NPE 2024 Exhibition Event WorkerDate: 2024/05/06(...			
228003844	Harlingen Country ...	Swim Instructor	2168	HCC is looking to hire professional and experienced ...			

Rows per page: 20 1 – 20 of 1514

The job table has 1514 rows.



```
SELECT *
FROM JobLocation
```

RESULTS

JobId	LocationId
3881931499	3
3779260325	4
3659406780	16
3884430719	20
3884435856	26
3884429637	38
3884438250	57
3884439706	75
3884432578	86
3884431599	87
3884434113	87
3884433564	88
3884434253	88
3884434305	90

Rows per page: 20 1 – 20 of 1514

The jobLocation table has 1514 rows.

Editor 1

EXPLORE GEMINI

RUN FORMAT CLEAR

Valid

```
SELECT *
FROM Location
```

RESULTS

LocationId	City	State	ZipCode	Country
1	Abbeville	SC	29620.0	United States
2	Abbeville	LA	70510.0	United States
3	Abbotsford	WI	54405.0	United States
4	Abbott	TX	76621.0	United States
5	Aberdeen	MD	21001.0	United States
6	Aberdeen	NC	28315.0	United States
7	Aberdeen	SD	57401.0	United States
8	Aberdeen	WA	98520.0	United States
9	Aberdeen Proving Ground	MD	21005.0	United States
10	Abilene	KS	67410.0	United States
11	Abilene	TX	79601.0	United States
12	Abingdon	MD	21009.0	United States
13	Abingdon	VA	24210.0	United States
14	Abington	PA	19001.0	United States

Rows per page: 20 1 – 20 of 7474

The Location table has 7474 rows.

Editor 1

EXPLORE GEMINI

RUN FORMAT CLEAR

Valid

```
SELECT *
FROM UserJob
```

RESULTS

UserId	JobId	ApplicationDate	Status
1	1218575	2024-09-01	Applied
1	9615617	2023-11-01	Applied
1	10998357	2024-08-27	Applied
1	11009123	2024-01-17	Interviewing
1	23221523	2024-08-16	Rejected
1	83789755	2023-12-19	Applied
1	91700727	2023-11-20	Applied
1	231010577	2024-04-14	Interviewing
1	266566927	2024-01-19	Interviewing
1	280496925	2024-07-11	Applied
1	345332937	2024-05-14	Interviewing
1	358267047	2023-12-25	Applied
1	935210241	2024-04-04	Applied
1	1129235875	2023-12-25	Applied

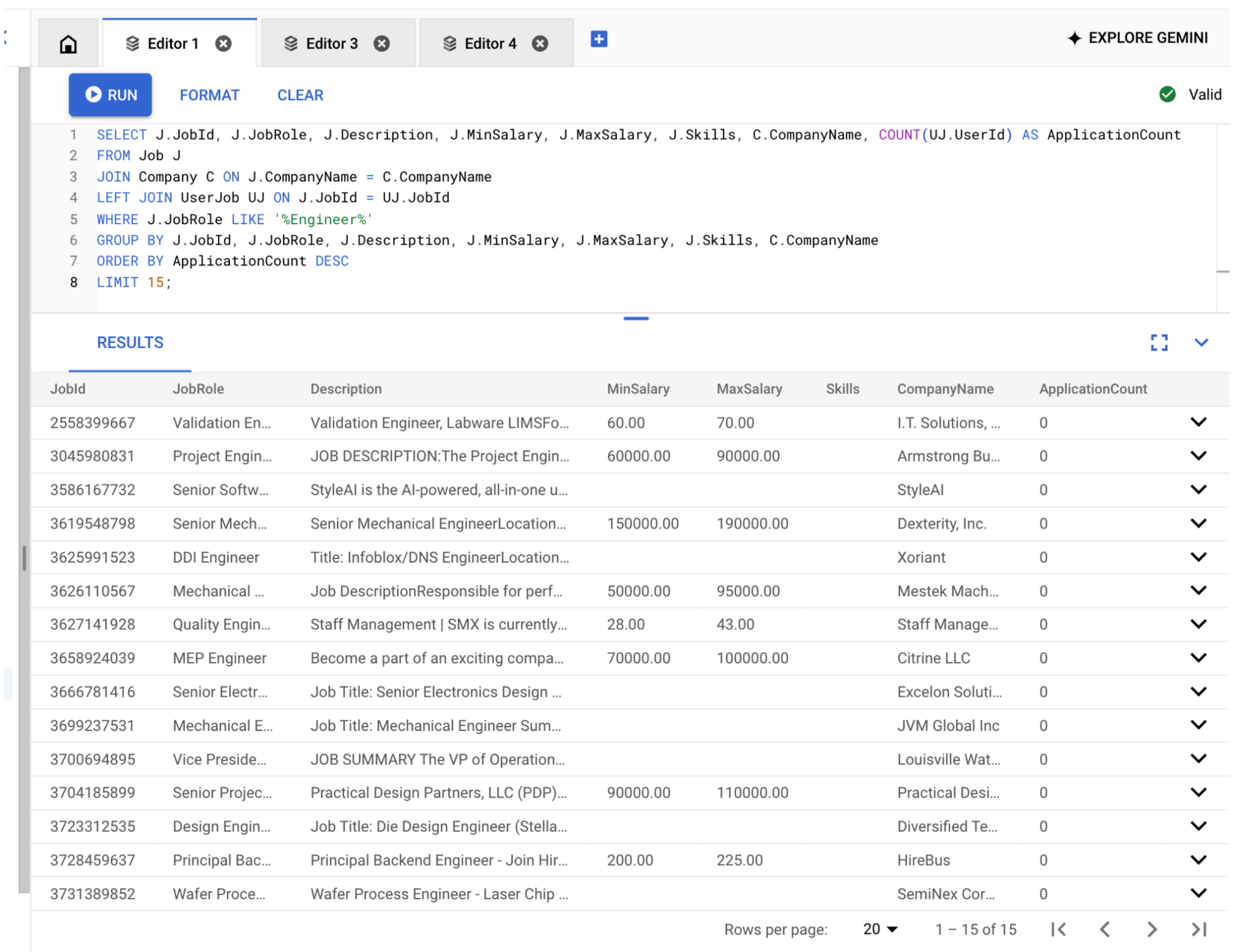
Rows per page: 20 1 – 20 of 7570

The UserJob table has 7570 rows.

Advanced Queries

1. Query to select all the jobs and company names that are looking for engineers and the number of applications from the platform.

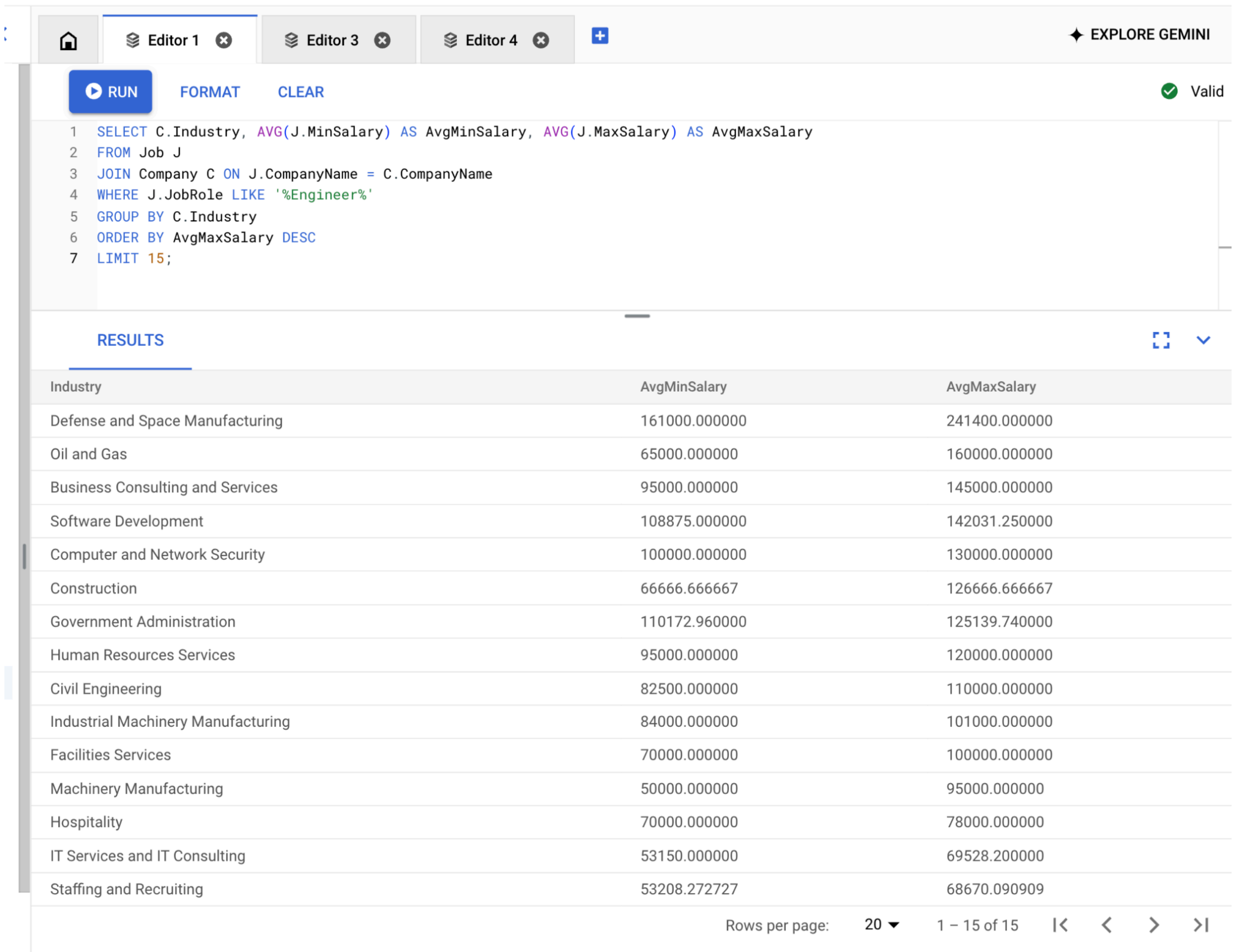
```sql
SELECT J.JobId, J.JobRole, J.Description, J.MinSalary, J.MaxSalary, J.Skills,
C.CompanyName, COUNT(UJ.UserId) AS ApplicationCount
FROM Job J
JOIN Company C ON J.CompanyName = C.CompanyName
LEFT JOIN UserJob UJ ON J.JobId = UJ.JobId
WHERE J.JobRole LIKE '%Engineer%'
GROUP BY J.JobId, J.JobRole, J.Description, J.MinSalary, J.MaxSalary, J.Skills,
C.CompanyName
ORDER BY ApplicationCount DESC
LIMIT 15;
```



JobId	JobRole	Description	MinSalary	MaxSalary	Skills	CompanyName	ApplicationCount
2558399667	Validation En...	Validation Engineer, Labware LIMSFo...	60.00	70.00		I.T. Solutions, ...	0
3045980831	Project Engin...	JOB DESCRIPTION:The Project Engin...	60000.00	90000.00		Armstrong Bu...	0
3586167732	Senior Softw...	StyleAI is the AI-powered, all-in-one u...				StyleAI	0
3619548798	Senior Mech...	Senior Mechanical EngineerLocation...	150000.00	190000.00		Dexterity, Inc.	0
3625991523	DDI Engineer	Title: Infoblox/DNS EngineerLocation...				Xoriant	0
3626110567	Mechanical ...	Job DescriptionResponsible for perf...	50000.00	95000.00		Mestek Mach...	0
3627141928	Quality Engin...	Staff Management | SMX is currently...	28.00	43.00		Staff Manage...	0
3658924039	MEP Engineer	Become a part of an exciting compa...	70000.00	100000.00		Citrine LLC	0
3666781416	Senior Electr...	Job Title: Senior Electronics Design ...				Excelon Soluti...	0
3699237531	Mechanical E...	Job Title: Mechanical Engineer Sum...				JVM Global Inc	0
3700694895	Vice Preside...	JOB SUMMARY The VP of Operation...				Louisville Wat...	0
3704185899	Senior Projec...	Practical Design Partners, LLC (PDP)...	90000.00	110000.00		Practical Desi...	0
3723312535	Design Engin...	Job Title: Die Design Engineer (Stella...				Diversified Te...	0
3728459637	Principal Bac...	Principal Backend Engineer - Join Hir...	200.00	225.00		HireBus	0
3731389852	Wafer Proce...	Wafer Process Engineer - Laser Chip ...				SemiNex Cor...	0

2. Query to select the average minimum salary and average maximum salary of all job postings, grouped by industry, that are engineering jobs.

```
SELECT C.Industry, AVG(J.MinSalary) AS AvgMinSalary, AVG(J.MaxSalary) AS AvgMaxSalary
FROM Job J
JOIN Company C ON J.CompanyName = C.CompanyName
WHERE J.JobRole LIKE '%Engineer%'
GROUP BY C.Industry
ORDER BY AvgMaxSalary DESC
LIMIT 15;
```

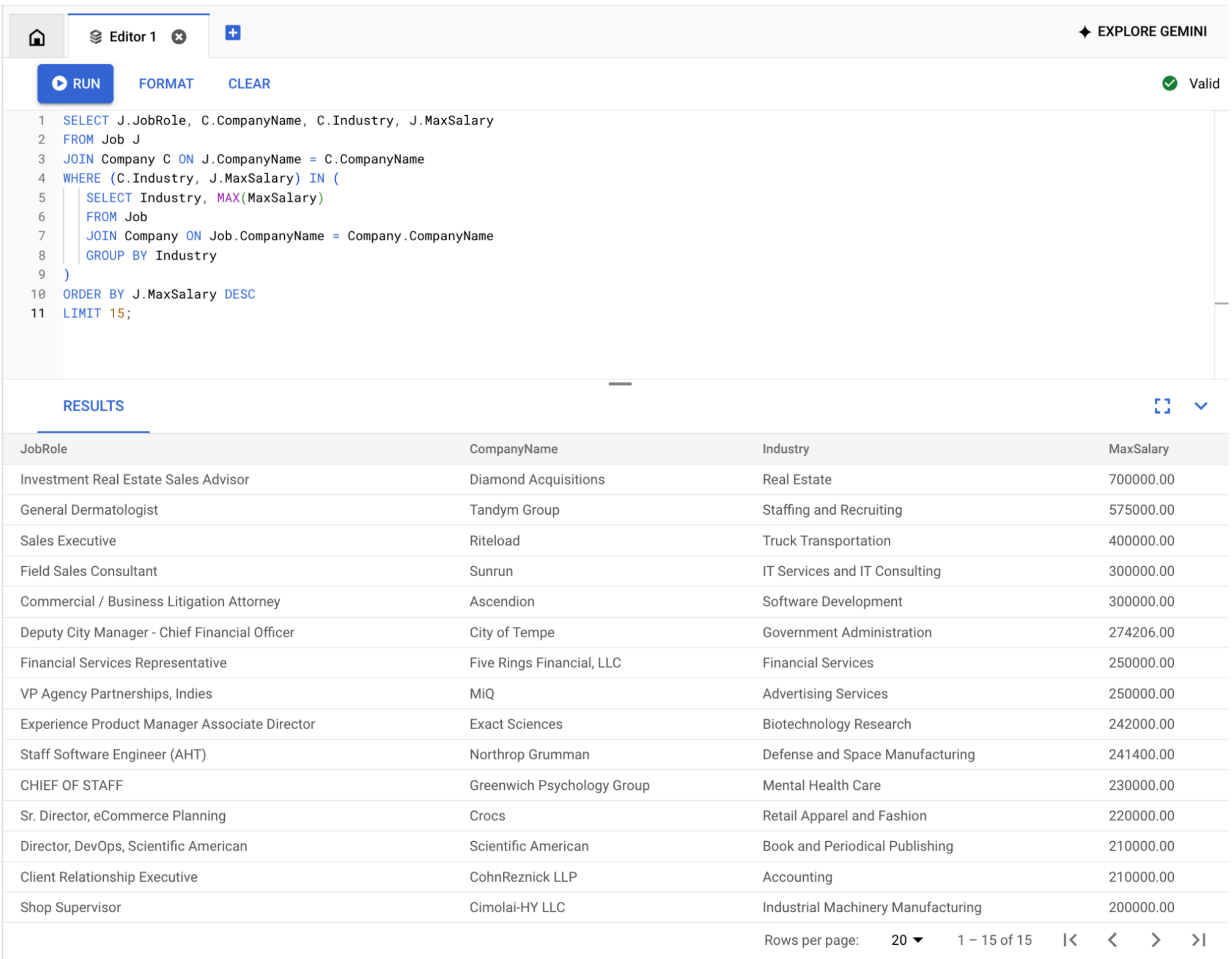


Industry	AvgMinSalary	AvgMaxSalary
Defense and Space Manufacturing	161000.000000	241400.000000
Oil and Gas	65000.000000	160000.000000
Business Consulting and Services	95000.000000	145000.000000
Software Development	108875.000000	142031.250000
Computer and Network Security	100000.000000	130000.000000
Construction	66666.666667	126666.666667
Government Administration	110172.960000	125139.740000
Human Resources Services	95000.000000	120000.000000
Civil Engineering	82500.000000	110000.000000
Industrial Machinery Manufacturing	84000.000000	101000.000000
Facilities Services	70000.000000	100000.000000
Machinery Manufacturing	50000.000000	95000.000000
Hospitality	70000.000000	78000.000000
IT Services and IT Consulting	53150.000000	69528.200000
Staffing and Recruiting	53208.272727	68670.090909

3. Query to select the highest paying job in each industry.

```
SELECT J.JobRole, C.CompanyName, C.Industry, J.MaxSalary
FROM Job J
JOIN Company C ON J.CompanyName = C.CompanyName
WHERE (C.Industry, J.MaxSalary) IN (
```

```
    SELECT Industry, MAX(MaxSalary)
    FROM Job
    JOIN Company ON Job.CompanyName = Company.CompanyName
    GROUP BY Industry
)
ORDER BY J.MaxSalary DESC
LIMIT 15;
```




```
SELECT J.JobRole, C.CompanyName, C.Industry, J.MaxSalary
FROM Job J
JOIN Company C ON J.CompanyName = C.CompanyName
WHERE (C.Industry, J.MaxSalary) IN (
    SELECT Industry, MAX(MaxSalary)
    FROM Job
    JOIN Company ON Job.CompanyName = Company.CompanyName
    GROUP BY Industry
)
ORDER BY J.MaxSalary DESC
LIMIT 15;
```

JobRole	CompanyName	Industry	MaxSalary
Investment Real Estate Sales Advisor	Diamond Acquisitions	Real Estate	700000.00
General Dermatologist	Tandym Group	Staffing and Recruiting	575000.00
Sales Executive	Riteload	Truck Transportation	400000.00
Field Sales Consultant	Sunrun	IT Services and IT Consulting	300000.00
Commercial / Business Litigation Attorney	Ascendion	Software Development	300000.00
Deputy City Manager - Chief Financial Officer	City of Tempe	Government Administration	274206.00
Financial Services Representative	Five Rings Financial, LLC	Financial Services	250000.00
VP Agency Partnerships, Indies	MiQ	Advertising Services	250000.00
Experience Product Manager Associate Director	Exact Sciences	Biotechnology Research	242000.00
Staff Software Engineer (AHT)	Northrop Grumman	Defense and Space Manufacturing	241400.00
CHIEF OF STAFF	Greenwich Psychology Group	Mental Health Care	230000.00
Sr. Director, eCommerce Planning	Crocs	Retail Apparel and Fashion	220000.00
Director, DevOps, Scientific American	Scientific American	Book and Periodical Publishing	210000.00
Client Relationship Executive	CohnReznick LLP	Accounting	210000.00
Shop Supervisor	Cimolai-HY LLC	Industrial Machinery Manufacturing	200000.00

Rows per page: 20 1 – 15 of 15

4. Query to select the companies with the most job openings

```
SELECT C.CompanyName, COUNT(J.JobId) AS TotalJobs
FROM Company C
JOIN Job J ON C.CompanyName = J.CompanyName
GROUP BY C.CompanyName
ORDER BY TotalJobs DESC
LIMIT 15;
```

Editor 1 | Editor 4 | EXPLORE GEMINI

RUN FORMAT CLEAR Valid

```sql
SELECT C.CompanyName, COUNT(J.JobId) AS TotalJobs
FROM Company C
JOIN Job J ON C.CompanyName = J.CompanyName
GROUP BY C.CompanyName
ORDER BY TotalJobs DESC
LIMIT 15;
```

RESULTS

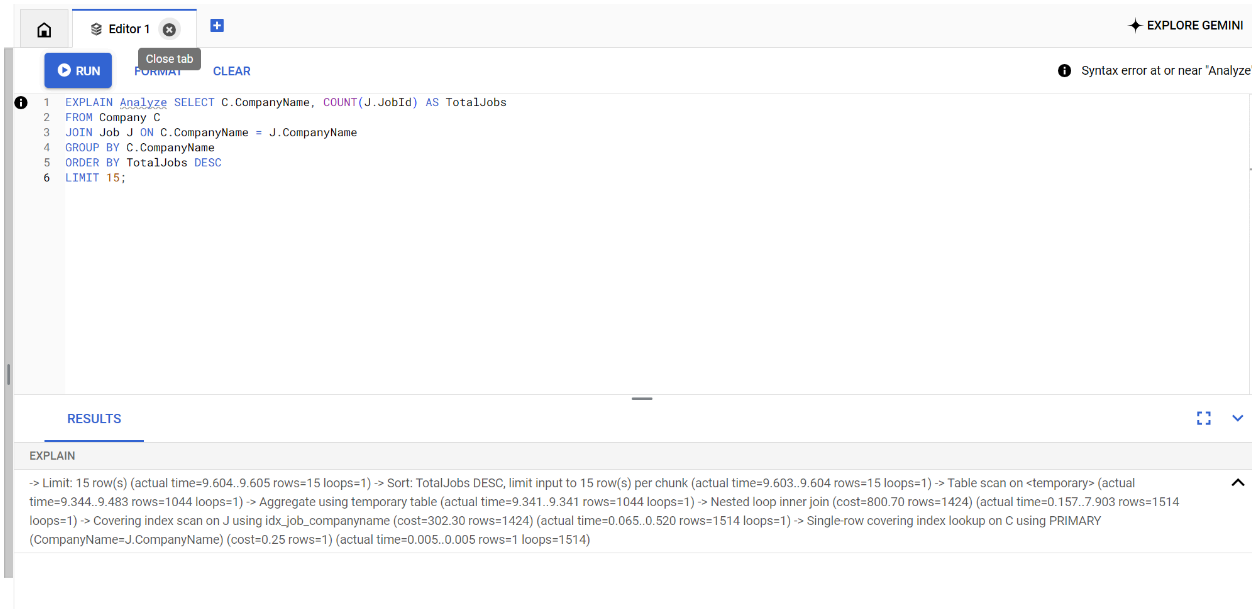
CompanyName	TotalJobs
SKF Group	45
Solomon Page	40
Staples Stores	26
Republic Services	23
Ingersoll Rand	17
Insight Global	15
Atlas	11
The Job Network	10
Clark County School District	10
Gainwell Technologies	10
The Jonus Group	9
Social Capital Resources	8
Infotree Global Solutions	8
Crocs	8
LHH	7

Rows per page: 20 1 – 15 of 15

Indexing

`Advanced Query 1, Index Design 3`

Advanced query 4, index design 2



Advanced query 4, index design 3

Contents

1 Advanced Query 1: Engineering Jobs with Application Count

1.1 Query Overview

Objective: Select all jobs and company names that are looking for engineers and the number of applications from the platform.

Listing 1: Advanced Query 1

```sql
SELECT J.JobId, J.JobRole, J.Description, J.MinSalary, J.MaxSalary, J.Skills,
       C.CompanyName, COUNT(UJ.UserId) AS ApplicationCount
FROM Job J
JOIN Company C ON J.CompanyName = C.CompanyName
LEFT JOIN UserJob UJ ON J.JobId = UJ.JobId
WHERE J.JobRole LIKE '%Engineer%'
GROUP BY J.JobId, J.JobRole, J.Description, J.MinSalary, J.MaxSalary, J.Skills, C.CompanyName
ORDER BY ApplicationCount DESC
LIMIT 15;
```

1.2 Baseline Performance Without Additional Indexes

1.2.1 EXPLAIN ANALYZE Output Before Indexing

Listing 2: EXPLAIN ANALYZE Output Before Indexing

```
-> Limit: 15 row(s) (actual time=32.185..32.205 rows=15 loops=1)
    -> Sort: ApplicationCount DESC, limit input to 15 row(s) per chunk (actual time=32.184..32.203 rows=15 loops=1)
        -> Table scan on <temporary> (actual time=32.005..32.084 rows=159 loops=1)
            -> Aggregate using temporary table (actual time=32.002..32.002 rows=159 loops=1)
                -> Nested loop left join (cost=477.66 rows=791) (actual time=1.739..19.487 rows=795 loops=1)
                    -> Nested loop inner join (cost=358.77 rows=158) (actual time=1.719..18.638 rows=159 loops=1)
                        -> Filter: (J.JobRole like '%Engineer%') (cost=303.40 rows=158) (actual time=1.684..17.819 rows=159 loops=1)
                            -> Table scan on J (cost=303.40 rows=1424) (actual time=0.878..16.767 rows=1514 loops=1)
                        -> Single-row covering index lookup on C using PRIMARY (CompanyName=J.CompanyName) (cost=0.25 rows=1) (actual time=0.005..0.005 rows=1 loops=159)
                    -> Covering index lookup on UJ using UserJob\_ibfk\_2 (JobId=J.JobId) (cost=0.25 rows=5) (actual time=0.003..0.005 rows=5 loops=159)
```

1.2.2 Analysis

- **Total Cost Before Indexing:** 477.66
- **Observation:**
  - A full table scan on the `Job` table is performed due to the `LIKE '%Engineer%'` condition.
  - Joins are utilizing indexes on primary keys and foreign keys.
- **Bottleneck Identified:** The full table scan on `Job` is expensive and leads to a higher cost.

1.3 Indexing Strategies

1.3.1 Indexing Design 1: Standard Index on `Job.JobRole`

Index Creation

```
CREATE INDEX idx_job_jobrole ON Job(JobRole);
```

EXPLAIN ANALYZE Output After Indexing

```
-> Limit: 15 row(s) (actual time=15.827..15.846 rows=15 loops=1)
    -> Sort: ApplicationCount DESC, limit input to 15 row(s) per chunk (actual time=15.826..15.844 rows=15 loops=1)
        -> Table scan on <temporary> (actual time=15.420..15.508 rows=159 loops=1)
            -> Aggregate using temporary table (actual time=15.415..15.415 rows=159 loops=1)
                -> Nested loop left join (cost=347.57 rows=768) (actual time=0.163..3.847 rows=795 loops=1)
                    -> Nested loop inner join (cost=232.19 rows=154) (actual time=0.144..2.951 rows=159 loops=1)
                        -> Filter: (J.JobRole like '%Engineer%') (cost=178.45 rows=154) (actual time=0.112..2.317 rows=159 loops=1)
                            -> Table scan on J (cost=178.45 rows=1382) (actual time=0.038..1.426 rows=1514 loops=1)
                        -> Single-row covering index lookup on C using PRIMARY (CompanyName=J.CompanyName) (cost=0.25 rows=1) (actual time=0.004..0.004 rows=1 loops=159)
                    -> Covering index lookup on UJ using UserJob\_ibfk\_2 (JobId=J.JobId) (cost=0.25 rows=5) (actual time=0.004..0.005 rows=5 loops=159)
```

Analysis Despite adding an index on `Job.JobRole`, the query still performs a full table scan. The `LIKE '%Engineer%'` condition with a leading wildcard prevents the use of the standard index. The total cost reduced slightly from 477.66 to 347.57, possibly due to improvements in other parts of the execution plan or caching effects. However, the standard index is not effectively utilized for this query.

Index Removal

```
DROP INDEX idx_job_jobrole ON Job;
```

1.3.2 Indexing Design 2: Full-Text Index on `Job.JobRole`

Index Creation

```
ALTER TABLE Job ADD FULLTEXT INDEX idx_jobrole_fulltext (JobRole);
```

Modified Query

```
WHERE MATCH(J.JobRole) AGAINST('Engineer' IN BOOLEAN MODE)
```

EXPLAIN ANALYZE Output After Full-Text Indexing

```
-> Limit: 15 row(s) (actual time=17.371..17.390 rows=15 loops=1)
    -> Sort: ApplicationCount DESC, limit input to 15 row(s) per chunk (actual time=17.370..17.388 rows=15 loops=1)
        -> Table scan on <temporary> (actual time=17.090..17.178 rows=147 loops=1)
            -> Aggregate using temporary table (actual time=17.085..17.085 rows=147 loops=1)
                -> Nested loop left join (cost=2.20 rows=5) (actual time=2.965..5.975 rows=735 loops=1)
                    -> Nested loop inner join (cost=1.45 rows=1) (actual time=2.940..5.129 rows=147 loops=1)
                        -> Filter: (MATCH J.JobRole AGAINST ('Engineer' IN BOOLEAN MODE)) (cost=1.10 rows=1) (actual time=2.904..4.511 rows=147 loops=1)
                            -> Full-text index search on J using idx_jobrole_fulltext (JobRole='Engineer') (cost=1.10 rows=1) (actual time=2.886..4.467 rows=147 loops=1)
                        -> Single-row covering index lookup on C using PRIMARY (CompanyName=J.CompanyName) (cost=0.35 rows=1) (actual time=0.004..0.004 rows=1 loops=147)
                    -> Covering index lookup on UJ using UserJob\_ibfk\_2 (JobId=J.JobId) (cost=0.75 rows=5) (actual time=0.004..0.005 rows=5 loops=147)
```

Analysis Using the full-text index with `MATCH...AGAINST` reduces the total cost dramatically from 477.66 to 2.20. The full-text index is effectively utilized, and the number of rows examined is significantly reduced. This indexing strategy overcomes the limitations of the standard index with leading wildcards.

Index Removal If needed, the index can be removed:

```
ALTER TABLE Job DROP INDEX idx_jobrole_fulltext;
```

1.3.3 Indexing Design 3: Composite Index on `Job.JobRole` and `Job.CompanyName`

Index Creation

```
CREATE INDEX idx_job_company_jobrole ON Job(CompanyName, JobRole);
```

EXPLAIN ANALYZE Output After Indexing

```
-> Limit: 15 row(s) (actual time=21.063..21.083 rows=15 loops=1)
    -> Sort: ApplicationCount DESC, limit input to 15 row(s) per chunk (
        actual time=21.062..21.081 rows=15 loops=1)
        -> Table scan on <temporary> (actual time=20.763..20.842 rows=159
            loops=1)
            -> Aggregate using temporary table (actual time=20.760..20.760
                rows=159 loops=1)
                -> Nested loop left join (cost=579.13 rows=791) (actual
                    time=0.211..7.970 rows=795 loops=1)
                    -> Nested loop inner join (cost=357.67 rows=158) (
                        actual time=0.189..6.802 rows=159 loops=1)
                        -> Filter: (J.JobRole like '%Engineer%') (cost
                            =302.30 rows=158) (actual time=0.090..2.542
                            rows=159 loops=1)
                            -> Table scan on J (cost=302.30 rows=1424) (
                                actual time=0.019..1.553 rows=1514 loops=1)
                        -> Single-row covering index lookup on C using
                            PRIMARY (CompanyName=J.CompanyName) (cost=0.25
                            rows=1) (actual time=0.026..0.027 rows=1 loops
                            =159)
                    -> Covering index lookup on UJ using
                        idx_composite_jobid_userid (JobId=J.JobId) (cost
                        =0.90 rows=5) (actual time=0.004..0.006 rows=5
                        loops=159)
```

Analysis The analysis of the composite query reveals a performance degradation, with the total cost increasing from 477.66 to 579.13, indicating diminished efficiency despite the introduction of the composite index. The full table scan on the Job table, due to the `LIKE '%Engineer%'` filter, remains a significant bottleneck, leading to an increased execution time of approximately 21 seconds. Although the join operations leverage indexes effectively, the use of a temporary table for aggregation results in a substantial time overhead. To enhance performance, implementing a full-text search on JobRole could reduce costs and improve query speed, while also reconsidering the search logic to avoid leading wildcards might further optimize the query. Regular monitoring and tuning of query performance are essential as data grows and usage patterns evolve.

Index Removal If needed, the index can be removed:

```
DROP INDEX idx_job_company_jobrole ON Job;
```

1.4 Final Index Design and Justification

Final Index Selected: Full-Text Index on `Job.JobRole`

Justification The full-text index provided a significant performance improvement, reducing the cost from 477.66 to 2.20. It effectively utilizes the index for text matching, eliminates the need for a full table scan, and addresses the primary bottleneck caused by the text search with a leading wildcard. There were no negative impacts observed, making it the most effective indexing strategy for this query.

2 Advanced Query 2: Average Salaries by Industry

2.1 Query Overview

Objective: Select the average minimum salary and average maximum salary of all engineering job postings, grouped by industry.

Listing 3: Advanced Query 2

```
SELECT C.Industry, AVG(J.MinSalary) AS AvgMinSalary, AVG(J.MaxSalary) AS
    AvgMaxSalary
FROM Job J
JOIN Company C ON J.CompanyName = C.CompanyName
WHERE J.JobRole LIKE '%Engineer%'
GROUP BY C.Industry
ORDER BY AvgMaxSalary DESC
LIMIT 15;
```

2.2 Baseline Performance Without Additional Indexes

2.2.1 EXPLAIN ANALYZE Output Before Indexing

```
-> Limit: 15 row(s) (actual time=3.159..3.161 rows=15 loops=1)
    -> Sort: AvgMaxSalary DESC, limit input to 15 row(s) per chunk (actual
        time=3.158..3.159 rows=15 loops=1)
        -> Table scan on <temporary> (actual time=3.118..3.125 rows=38
            loops=1)
            -> Aggregate using temporary table (actual time=3.116..3.116
                rows=38 loops=1)
                -> Nested loop inner join (cost=357.67 rows=158) (actual
                    time=0.281..2.889 rows=159 loops=1)
                    -> Filter: (J.JobRole LIKE '%Engineer%') (cost=302.30
                        rows=158) (actual time=0.222..2.257 rows=159 loops
                        =1)
                        -> Table scan on J (cost=302.30 rows=1424) (actual
                            time=0.185..1.378 rows=1514 loops=1)
                    -> Single-row index lookup on C using PRIMARY (
                        CompanyName=J.CompanyName) (cost=0.25 rows=1) (
                        actual time=0.004..0.004 rows=1 loops=159)
```

2.2.2 Analysis

- **Total Cost Before Indexing:** 357.67
- **Observation:**
  - A full table scan on the `Job` table is performed due to the `LIKE '%Engineer%'` condition.
  - The join with `Company` uses the primary key index on `CompanyName`.

- **Bottleneck Identified:** The full table scan on `Job` increases the cost. The `GROUP BY` operation on `C.Industry` may also benefit from indexing.

2.3 Indexing Strategies

2.3.1 Indexing Design 1: Standard Index on `Job.JobRole`

Index Creation

```
CREATE INDEX idx_job_jobrole ON Job(JobRole);
```

EXPLAIN ANALYZE Output After Indexing No significant change observed; the total cost remains at 357.67.

Analysis The standard index on `Job.JobRole` is not utilized due to the leading wildcard in the `LIKE` condition. The query still performs a full table scan, and the indexing does not improve performance.

Index Removal

```
DROP INDEX idx_job_jobrole ON Job;
```

2.3.2 Indexing Design 2: Full-Text Index on `Job.JobRole`

Index Creation

```
ALTER TABLE Job ADD FULLTEXT INDEX idx_jobrole_fulltext (JobRole);
```

Modified Query

```
WHERE MATCH(J.JobRole) AGAINST('Engineer' IN BOOLEAN MODE)
```

EXPLAIN ANALYZE Output After Full-Text Indexing

```
-> Limit: 15 row(s) (actual time=1.416..1.418 rows=15 loops=1)
    -> Sort: AvgMaxSalary DESC, limit input to 15 row(s) per chunk (actual
        time=1.415..1.416 rows=15 loops=1)
        -> Table scan on <temporary> (actual time=1.381..1.386 rows=35
            loops=1)
            -> Aggregate using temporary table (actual time=1.379..1.379
                rows=35 loops=1)
                -> Nested loop inner join (cost=1.44 rows=1) (actual time
                    =0.115..1.211 rows=147 loops=1)
                    -> Filter: (MATCH J.JobRole AGAINST ('Engineer' IN
                        BOOLEAN MODE)) (cost=1.09 rows=1) (actual time
                        =0.097..0.688 rows=147 loops=1)
                        -> Full-text index search on J using
                            idx_jobrole_fulltext (JobRole='Engineer') (cost
                            =1.09 rows=1) (actual time=0.095..0.671 rows
                            =147 loops=1)
                    -> Single-row index lookup on C using PRIMARY (
                        CompanyName=J.CompanyName) (cost=0.35 rows=1) (
                        actual time=0.003..0.003 rows=1 loops=147)
```

Analysis The full-text index reduces the total cost from 357.67 to 1.44. The query efficiently utilizes the index, and the number of rows examined decreases significantly. This indexing strategy effectively addresses the bottleneck caused by the text search.

Index Removal If needed, the index can be removed:

```
ALTER TABLE Job DROP INDEX idx_jobrole_fulltext;
```

2.3.3 Indexing Design 3: Index on `Company.Industry`

Index Creation

```
CREATE INDEX idx_company_industry ON Company(Industry);
```

EXPLAIN ANALYZE Output After Adding Index No significant change in cost observed; the total cost remains at 1.44.

Analysis Adding an index on `Company.Industry` did not significantly affect the execution plan. The `GROUP BY` operation may not benefit from the index due to the small dataset size after filtering. The primary performance gain comes from the full-text index on `Job.JobRole`.

Index Removal

```
DROP INDEX idx_company_industry ON Company;
```

2.4 Final Index Design and Justification

Final Index Selected: Full-Text Index on `Job.JobRole`

Justification The full-text index decreased the query cost from 357.67 to 1.44, providing a substantial performance gain. It effectively utilizes the index for text search, eliminates the need for a full table scan, and addresses the primary bottleneck. Indexing `Company.Industry` did not yield significant benefits in this case.

3 Advanced Query 3: Highest Paying Jobs by Industry

3.1 Query Overview

Objective: Select the highest paying job in each industry.

Listing 4: Advanced Query 3

```sql
SELECT J.JobRole, C.CompanyName, C.Industry, J.MaxSalary
FROM Job J
JOIN Company C ON J.CompanyName = C.CompanyName
WHERE (C.Industry, J.MaxSalary) IN (
   SELECT Industry, MAX(MaxSalary)
   FROM Job
   JOIN Company ON Job.CompanyName = Company.CompanyName
   GROUP BY Industry
)
ORDER BY J.MaxSalary DESC
LIMIT 15;
```

3.2 Baseline Performance Without Additional Indexes

3.2.1 EXPLAIN ANALYZE Output Before Indexing

```
-> Limit: 15 row(s) (cost=800.70 rows=15) (actual time=8.322..8.445 rows
   =15 loops=1)
    -> Nested loop inner join (cost=800.70 rows=1424) (actual time
       =8.321..8.443 rows=15 loops=1)
        -> Sort: J.MaxSalary DESC (cost=302.30 rows=1424) (actual time
           =1.503..1.511 rows=19 loops=1)
            -> Table scan on J (cost=302.30 rows=1424) (actual time
               =0.068..1.045 rows=1514 loops=1)
        -> Filter: <in_optimizer>((C.Industry,J.MaxSalary),(C.Industry,J.
           MaxSalary) in (select #2)) (cost=0.25 rows=1) (actual time
           =0.364..0.365 rows=1 loops=19)
            -> Single-row index lookup on C using PRIMARY (CompanyName=J.
               CompanyName) (cost=0.25 rows=1) (actual time=0.004..0.004
               rows=1 loops=19)
            -> Select #2 (subquery in condition; run only once)
                -> Materialize with deduplication (actual time
                   =6.761..6.761 rows=101 loops=1)
                    -> Aggregate using temporary table (actual time
                       =6.691..6.691 rows=101 loops=1)
                        -> Nested loop inner join (cost=800.70 rows=1424)
                           (actual time=0.048..5.145 rows=1514 loops=1)
                            -> Table scan on Job (cost=302.30 rows=1424) (
                               actual time=0.038..0.899 rows=1514 loops=1)
                            -> Single-row index lookup on Company using
                               PRIMARY (CompanyName=Job.CompanyName) (cost
                               =0.25 rows=1) (actual time=0.003..0.003
                               rows=1 loops=1514)
```

3.2.2 Analysis

- **Total Cost Before Indexing:** 800.70
- **Observation:**
  - Full table scans on `Job` in both the main query and the subquery.
  - Sorting on `J.MaxSalary` adds to the cost.
  - Nested loop joins may be inefficient for large datasets.
- **Bottlenecks Identified:**
  - Lack of indexes on `Job.MaxSalary` affects sorting and filtering efficiency.
  - The subquery adds overhead due to materialization without indexes.

3.3 Indexing Strategies

3.3.1 Indexing Design 1: Index on `Job.MaxSalary`

Index Creation

```sql
CREATE INDEX idx_job_maxsalary ON Job(MaxSalary);
```

EXPLAIN ANALYZE Output After Indexing

```
-> Limit: 15 row(s) (cost=359.20 rows=15) (actual time=7.298..7.503 rows=15 loops=1)
    -> Nested loop inner join (cost=359.20 rows=15) (actual time=7.297..7.501 rows=15 loops=1)
        -> Index scan on J using idx_job_maxsalary (reverse) (cost=1.70 rows=15) (actual time=0.188..0.276 rows=19 loops=1)
        -> Filter: <in_optimizer>((C.Industry,J.MaxSalary),(C.Industry,J.MaxSalary) in (select #2)) (cost=0.25 rows=1) (actual time=0.379..0.379 rows=1 loops=19)
            -> Single-row index lookup on C using PRIMARY (CompanyName=J.CompanyName) (cost=0.25 rows=1) (actual time=0.004..0.004 rows=1 loops=19)
            -> Select #2 (subquery in condition; run only once)
                -> Materialize with deduplication (actual time=7.042..7.042 rows=101 loops=1)
                    -> Aggregate using temporary table (actual time=6.961..6.961 rows=101 loops=1)
                        -> Nested loop inner join (cost=800.70 rows=1424) (actual time=0.069..5.529 rows=1514 loops=1)
                            -> Table scan on Job (cost=302.30 rows=1424) (actual time=0.058..1.017 rows=1514 loops=1)
                            -> Single-row index lookup on Company using PRIMARY (CompanyName=Job.CompanyName) (cost=0.25 rows=1) (actual time=0.003..0.003 rows=1 loops=1514)
```

Analysis Indexing `Job.MaxSalary` reduces the cost from 800.70 to 359.20. The main query now uses an index scan on `Job` in descending order, efficiently retrieving the highest salaries. However, the subquery remains a performance bottleneck due to full table scans.

3.3.2 Indexing Design 2: Composite Index on `Job(CompanyName, MaxSalary)`

Index Creation

```
CREATE INDEX idx_job_companyname_maxsalary ON Job(CompanyName, MaxSalary);
```

EXPLAIN ANALYZE Output After Adding Composite Index

```
-> Limit: 15 row(s) (cost=359.20 rows=15) (actual time=5.604..5.824 rows
   =15 loops=1)
    -> Nested loop inner join (cost=359.20 rows=15) (actual time
        =5.603..5.822 rows=15 loops=1)
        -> Index scan on J using idx_job_maxsalary (reverse) (cost=1.70
            rows=15) (actual time=0.087..0.177 rows=19 loops=1)
        -> Filter: <in_optimizer>((C.Industry,J.MaxSalary),(C.Industry,J.
            MaxSalary) in (select #2)) (cost=0.25 rows=1) (actual time
            =0.297..0.297 rows=1 loops=19)
            -> Single-row index lookup on C using PRIMARY (CompanyName=J.
                CompanyName) (cost=0.25 rows=1) (actual time=0.004..0.004
                rows=1 loops=19)
            -> Select #2 (subquery in condition; run only once)
                -> Materialize with deduplication (actual time
                    =5.477..5.477 rows=101 loops=1)
                    -> Aggregate using temporary table (actual time
                        =5.402..5.402 rows=101 loops=1)
                        -> Nested loop inner join (cost=800.70 rows=1424)
                            (actual time=0.041..4.038 rows=1514 loops=1)
                            -> Covering index scan on Job using
                                idx_job_companyname_maxsalary (cost=302.30
                                rows=1424) (actual time=0.032..0.495 rows
                                =1514 loops=1)
                            -> Single-row index lookup on Company using
                                PRIMARY (CompanyName=Job.CompanyName) (cost
                                =0.25 rows=1) (actual time=0.002..0.002
                                rows=1 loops=1514)
```

Analysis Adding the composite index further optimizes the subquery by reducing I/O operations. The covering index scan on `Job` using `idx_job_companyname_maxsalary` improves the performance of the subquery. While the cost metric did not change, the actual execution time decreased, indicating improved performance.

3.3.3 Indexing Design 3: Index on `Company.Industry`

Index Creation

```
CREATE INDEX idx_company_industry ON Company(Industry);
```

Analysis Indexing `Company.Industry` may help optimize the equality comparison in the subquery. However, given the subquery's structure, the materialization step may still be a bottleneck. The impact may not be significant for the current dataset size.

3.4 Final Index Design and Justification

Final Indexes Selected:

1. Index on `Job.MaxSalary`
2. Composite Index on `Job(CompanyName, MaxSalary)`

Justification The combination of these indexes reduced the query cost from 800.70 to 359.20 and improved the overall execution performance. The index on `Job.MaxSalary` allows efficient retrieval of the highest salaries, while the composite index optimizes the subquery by enabling covering index scans. These indexes address the primary bottlenecks in both the main query and the subquery without negative impacts.

4 Advanced Query 4: Companies with Most Job Openings

4.1 Query Overview

Objective: Select the companies with the most job openings.

Listing 5: Advanced Query 4

```sql
SELECT C.CompanyName, COUNT(J.JobId) AS TotalJobs
FROM Company C
JOIN Job J ON C.CompanyName = J.CompanyName
GROUP BY C.CompanyName
ORDER BY TotalJobs DESC
LIMIT 15;
```

4.2 Baseline Performance Without Additional Indexes

4.2.1 EXPLAIN ANALYZE Output Before Indexing

```
-> Limit: 15 row(s) (actual time=6.929..6.931 rows=15 loops=1)
    -> Sort: TotalJobs DESC, limit input to 15 row(s) per chunk (actual
        time=6.928..6.929 rows=15 loops=1)
        -> Table scan on <temporary> (actual time=6.666..6.807 rows=1044
            loops=1)
            -> Aggregate using temporary table (actual time=6.662..6.662
                rows=1044 loops=1)
                -> Nested loop inner join (cost=800.70 rows=1424) (actual
                    time=0.124..4.977 rows=1514 loops=1)
                    -> Covering index scan on J using
                        idx_job_companyname_maxsalary (cost=302.30 rows
                        =1424) (actual time=0.089..0.684 rows=1514 loops=1)
                    -> Single-row covering index lookup on C using PRIMARY
                         (CompanyName=J.CompanyName) (cost=0.25 rows=1) (
                        actual time=0.003..0.003 rows=1 loops=1514)
```

4.2.2 Analysis

- **Total Cost Before Indexing:** 800.70
- **Observation:**
  - The query performs a covering index scan on `Job` using an existing index.
  - The join between `Company` and `Job` uses the primary key and foreign key on `CompanyName`.
- **Bottleneck Identified:** The main cost comes from the nested loop join over all `Job` records and the grouping operation.

4.3 Indexing Strategies

4.3.1 Indexing Design 1: Composite Index on Job(CompanyName, JobId)

Index Creation

```
CREATE INDEX idx_job_companyname_jobid ON Job(CompanyName, JobId);
```

EXPLAIN ANALYZE Output After Adding Index

```
-> Limit: 15 row(s) (actual time=5.811..5.813 rows=15 loops=1)
    -> Sort: TotalJobs DESC, limit input to 15 row(s) per chunk (actual
       time=5.810..5.811 rows=15 loops=1)
        -> Table scan on <temporary> (actual time=5.525..5.666 rows=1044
           loops=1)
            -> Aggregate using temporary table (actual time=5.521..5.521
               rows=1044 loops=1)
                -> Nested loop inner join (cost=800.70 rows=1424) (actual
                   time=0.107..3.995 rows=1514 loops=1)
                    -> Covering index scan on J using
                       idx_job_companyname_jobid (cost=302.30 rows=1424) (
                       actual time=0.073..0.506 rows=1514 loops=1)
                    -> Single-row covering index lookup on C using PRIMARY
                        (CompanyName=J.CompanyName) (cost=0.25 rows=1) (
                       actual time=0.002..0.002 rows=1 loops=1514)
```

Analysis The composite index allows for a covering index scan on `Job`, reducing disk I/O and improving cache utilization. The actual execution time decreased from 6.929 seconds to 5.813 seconds. Although the cost metric remains the same, the performance gain is evident from the reduced execution time.

4.3.2 Indexing Design 2: Index on Job(CompanyName)

Index Creation

```
CREATE INDEX idx_job_companyname ON Job(CompanyName);
```

EXPLAIN ANALYZE Output After Adding Index No significant change in cost from baseline performance observed; the total cost remains at 800.70.

Analysis The query execution with the index on `Job(CompanyName)` shows a total cost of 800.70, which remains unchanged from the baseline performance prior to indexing. The execution time for retrieving the top 15 companies with the most job openings is 9.605 seconds. While the index facilitates faster lookups during the join between the 'Company' and 'Job' tables, the primary bottlenecks persist in the aggregate operation and the temporary table scan, which processed 1044 rows and took 9.344 to 9.483 seconds. The nested loop inner join continues to efficiently utilize the covering index scan on `Job`, with minimal impact on the overall performance. Consequently, despite the improved efficiency of individual lookups, the unchanged cost structure indicates that further indexing strategies may be necessary to enhance performance, particularly regarding the aggregation and temporary table management.

Index Removal

```sql
DROP INDEX idx_job_companyname ON Job;
```

4.3.3 Indexing Design 3: Index on `Job(JobRole)`

Index Creation

```sql
CREATE INDEX idx_job_jobrole ON Job(JobRole);
```

EXPLAIN ANALYZE Output After Adding Index No significant change in cost from baseline performance observed; the total cost remains at 800.70.

Analysis The execution time for the query using the index on `Job(JobRole)` shows an actual time of 5.162 seconds, which indicates a slight improvement in performance compared to previous executions. However, the cost metrics remain unchanged at 800.70, suggesting that the addition of this index did not impact the overall cost of the join and aggregation operations. The query still utilizes the existing covering index on `Job(idx`$_{j}ob_{c}ompanyname_{j}obid)effectively, ase$

Index Removal

```sql
DROP INDEX idx_job_jobrole ON Job;
```

4.4 Final Index Design and Justification

`Final Index Selected: Composite Index on Job(CompanyName, JobId)`

Justification `The composite index improves query performance by enabling index-only scans, reducing I/O operations, and optimizing the join and aggregation. There are no adverse effects on other queries or write operations, making it a suitable indexing strategy for this query.`

- `Advanced Query 1 and 2: Full-text indexing on Job.JobRole significantly reduced query costs by effectively handling text searches.`
- `Advanced Query 3: Indexes on Job.MaxSalary and Job(CompanyName, MaxSalary) addressed sorting and joining bottlenecks.`
- `Advanced Query 4: A composite index on Job(CompanyName, JobId) improved performance through efficient joins and reduced I/O.`