

UniCareers

Database Implementation and Indexing

Keeron Huang, Libin Wang, Mingyan Gao, Yanzi Li¹

¹University of Illinois Urbana-Champaign

Keywords: Job Exploration, College Career, Education



1 | Modified ER Diagram

We do some modifications on our ER model of Stage2. Below is the new image of our ER model.

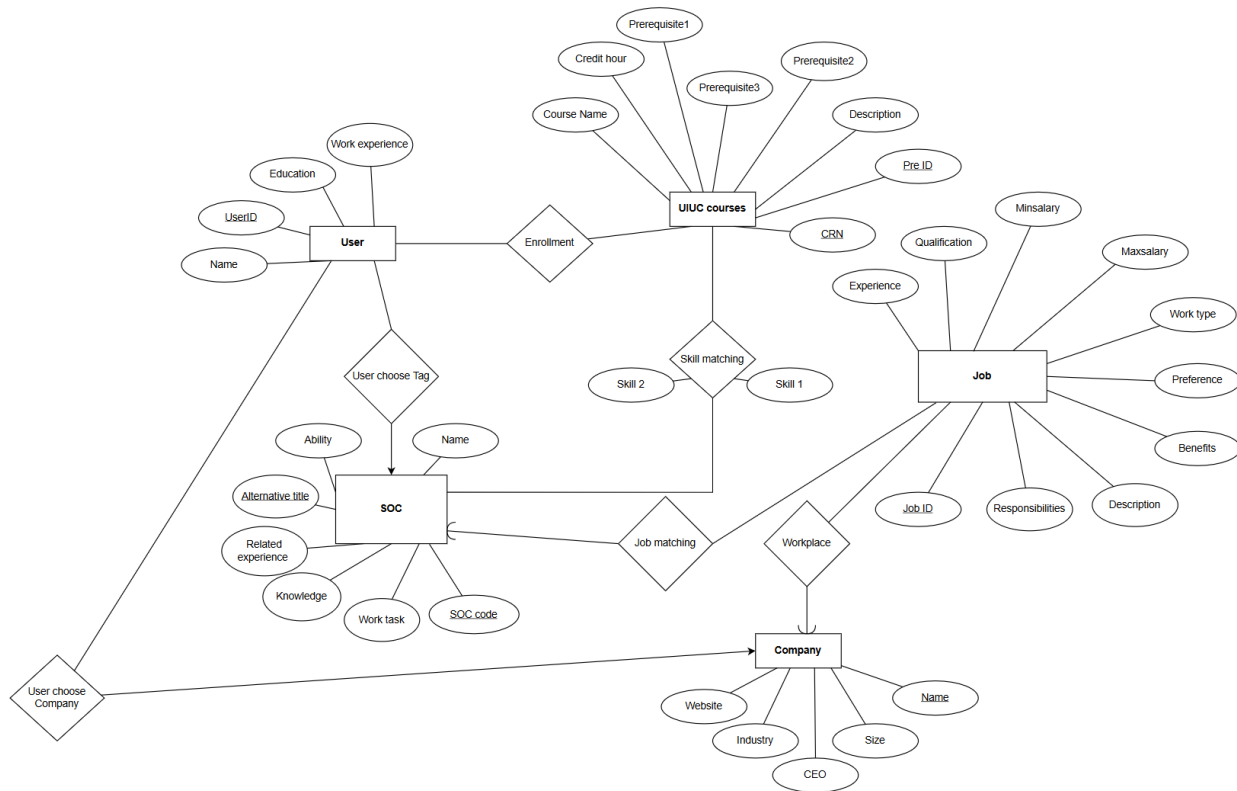
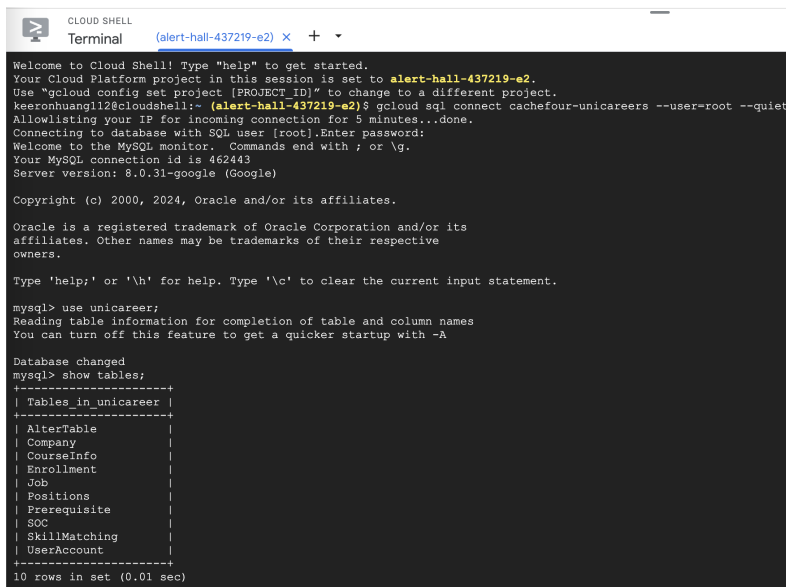


Figure 1: ER Model for Project Database

- **Type of relationship:** We switch the relationships between SOC and Job entities, between Company and Job entities from 'at most one' to 'exactly one'. Since after we observe our datasets, we found that every job can match exactly one SOC code and one company.
- **Change on Job:** We delete the original 'Job title' attribute from Job entity since its information overlaps with the attribute 'role'. Also, we want to provide the users with information about average salary of each company. So we split the original 'Salary Range' attribute into two parts: 'MinSalary' and 'MaxSalary'.
- **Change on SOC:** We delete the original 'Description' attribute from SOC entity since it is a very general concept. The detailed description of a SOC job is actually within other attributes like 'Work task', 'Related Experience', et al.

2 | Database implementation

2.1 Implementation of Tables on GCP



```

CLOUD SHELL
Terminal (alert-hall-437219-e2) X +
Welcome to Cloud Shell! Type "help" to get started.
Your Cloud Platform project in this session is set to alert-hall-437219-e2.
Use "gcloud config set project [PROJECT_ID]" to change to a different project.
keeronhuang12@cloudshell:~ (alert-hall-437219-e2) $ gcloud sql connect cachefour-unicareers --user=root --quiet
Allowlisting your IP for incoming connection for 5 minutes...done.
Connecting to database with SQL user [root].Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 462443
Server version: 8.0.31-google (Google)

Copyright (c) 2000, 2024, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> use unicareer;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> show tables;
+-----+
| Tables in unicareer |
+-----+
| AlterTable           |
| Company              |
| CourseInfo           |
| Enrollment           |
| Job                  |
| Positions            |
| Prerequisite          |
| SOC                  |
| SkillMatching        |
| UserAccount          |
+-----+
10 rows in set (0.01 sec)

```

Figure 2: GCP Implementation

2.2 Detailed DDL Commands

Below are DDL commands we used to create out relational tables, combined with default indices generated by MySQL Studio.

2.2.1 User Management

```

1 CREATE TABLE UserAccount (
2     UserId INT PRIMARY KEY,
3     Name VARCHAR(255) NOT NULL,
4     Experience INT,
5     Idealcom VARCHAR(255),
6     Education VARCHAR(255),
7     TagCode VARCHAR(255),
8     FOREIGN KEY (TagCode) REFERENCES SOC(SOC_code)
9         ON DELETE SET NULL,
10    FOREIGN KEY (Idealcom) REFERENCES Company(Name)
11        ON DELETE SET NULL
12 );

```

Listing 1: UserAccount Table

2.2.2 Standard Occupational Classification (SOC)

```

1 CREATE TABLE SOC (
2     SOC_code VARCHAR(255) PRIMARY KEY,
3     Name VARCHAR(255),
4     RelatedEx INT,

```

```

5     Knowledge VARCHAR(255),
6     Ability VARCHAR(255),
7     Worktask VARCHAR(511)
8 );

```

Listing 2: SOC Table

```

1 CREATE TABLE AlterTable (
2     SOCCode VARCHAR(255),
3     AlterTitle VARCHAR(255),
4     PRIMARY KEY (AlterTitle, SOCCode),
5     FOREIGN KEY (SOCCode) REFERENCES SOC(SOC_code)
6         ON DELETE CASCADE
7 );

```

Listing 3: AlterTable

2.2.3 Job Management

```

1 CREATE TABLE Job (
2     JobId VARCHAR(255) PRIMARY KEY,
3     Role VARCHAR(255),
4     AvgExperience INT,
5     Qualification VARCHAR(255),
6     MinSalary INT,
7     MaxSalary INT,
8     Type VARCHAR(255),
9     Preference VARCHAR(255),
10    Benefits VARCHAR(255),
11    CompanyName VARCHAR(255),
12    FOREIGN KEY (Role) REFERENCES AlterTable(AlterTitle)
13        ON DELETE CASCADE,
14    FOREIGN KEY (CompanyName) REFERENCES Company(Name)
15        ON DELETE SET NULL
16 );

```

Listing 4: Job Table

```

1 mysql> show indexes from Job;
2 +-----+-----+-----+-----+-----+
3 | Table | Non_unique | Key_name | Seq_in_index | Column_name |
4 +-----+-----+-----+-----+-----+
5 | Job   |          0 | PRIMARY |             1 | JobId       |
6 | Job   |          1 | Role    |             1 | Role        |
7 | Job   |          1 | CompanyName |             1 | CompanyName |
8 +-----+-----+-----+-----+-----+
9 3 rows in set (0.02 sec)

```

Listing 5: Default Indexes from Job

```

1 CREATE TABLE Positions (
2     Role VARCHAR(255) PRIMARY KEY,
3     Responsibilities VARCHAR(511),
4     Description VARCHAR(511),
5     FOREIGN KEY (Role) REFERENCES Job(Role)
6         ON DELETE CASCADE
7 );

```

Listing 6: Position Table

2.2.4 Company Information

```

1 CREATE TABLE Company (
2     Name VARCHAR(255) PRIMARY KEY,
3     Size INT,
4     CEO VARCHAR(255),
5     Industry VARCHAR(255),
6     Website VARCHAR(255)
7 );

```

Listing 7: Company Table

```

1 mysql> show indexes from Company;
2 +-----+-----+-----+-----+-----+
3 | Table | Non_unique | Key_name | Seq_in_index | Column_name |
4 +-----+-----+-----+-----+-----+
5 | Company | 0 | PRIMARY | 1 | Name |
6 +-----+-----+-----+-----+-----+
7 1 row in set (0.00 sec)

```

Listing 8: Default Indexes from Company

2.2.5 Course Management

```

1 CREATE TABLE CourseInfo (
2     CRN VARCHAR(255) PRIMARY KEY,
3     CourseName VARCHAR(255),
4     CreditHour INT,
5     Description VARCHAR(1023)
6 );

```

Listing 9: CourseInfo Table

```

1 mysql> show indexes from CourseInfo;
2 +-----+-----+-----+-----+-----+
3 | Table | Non_unique | Key_name | Seq_in_index | Column_name |
4 +-----+-----+-----+-----+-----+
5 | CourseInfo | 0 | PRIMARY | 1 | CRN |
6 +-----+-----+-----+-----+-----+
7 1 row in set (0.00 sec)

```

Listing 10: Default Indexes from CourseInfo

```

1 CREATE TABLE Prerequisite (
2     PreID INT,
3     CRN VARCHAR(255),
4     Pre1 VARCHAR(255),
5     Pre2 VARCHAR(255),
6     Pre3 VARCHAR(255),
7     PRIMARY KEY (PreID, CRN),
8     FOREIGN KEY (CRN) REFERENCES CourseInfo(CRN)
9     ON DELETE CASCADE
10 );

```

Listing 11: Prerequisite Table

```

1 mysql> show indexes from Prerequisite;
2 +-----+-----+-----+-----+-----+
3 | Table          | Non_unique | Key_name | Seq_in_index | Column_name |
4 +-----+-----+-----+-----+-----+
5 | Prerequisite |          0 | PRIMARY |          1 | PreID       |
6 | Prerequisite |          0 | PRIMARY |          2 | CRN         |
7 | Prerequisite |          1 | CRN     |          1 | CRN         |
8 +-----+-----+-----+-----+-----+
9 3 rows in set (0.00 sec)

```

Listing 12: Default Indexes from Prerequisite

2.2.6 Relationship Management

```

1 CREATE TABLE SkillMatching(
2     CRN VARCHAR(255),
3     SOCode VARCHAR(255),
4     Skill1 VARCHAR(255),
5     Skill2 VARCHAR(255),
6     PRIMARY KEY(CRN, SOCode),
7     FOREIGN KEY(CRN) REFERENCES CourseInfo(CRN)
8         ON DELETE CASCADE,
9     FOREIGN KEY(SOCode) REFERENCES SOC(SOC_code)
10        ON DELETE CASCADE
11 );

```

Listing 13: Skill-Matching Table

```

1 mysql> show indexes from SkillMatching;
2 +-----+-----+-----+-----+-----+
3 | Table          | Non_unique | Key_name | Seq_in_index | Column_name |
4 +-----+-----+-----+-----+-----+
5 | SkillMatching |          0 | PRIMARY |          1 | CRN         |
6 | SkillMatching |          0 | PRIMARY |          2 | SOCode      |
7 | SkillMatching |          1 | SOCode  |          1 | SOCode      |
8 +-----+-----+-----+-----+-----+
9 3 rows in set (0.02 sec)

```

Listing 14: Default Indexes from SkillMatching

```

1 CREATE TABLE Enrollment (
2     UserId INT,
3     CRN VARCHAR(255),
4     PRIMARY KEY (UserId, CRN),
5     FOREIGN KEY (UserId) REFERENCES UserAccount(UserId)
6         ON DELETE CASCADE,
7     FOREIGN KEY (CRN) REFERENCES CourseInfo(CRN)
8         ON DELETE CASCADE
9 );

```

Listing 15: Enrollment Table

```
1 mysql> show indexes from Enrollment;
2 +-----+-----+-----+-----+-----+
3 | Table      | Non_unique | Key_name | Seq_in_index | Column_name |
4 +-----+-----+-----+-----+-----+
5 | Enrollment |          0 | PRIMARY |             1 | UserId      |
6 | Enrollment |          0 | PRIMARY |             2 | CRN         |
7 | Enrollment |          1 | CRN     |             1 | CRN         |
8 +-----+-----+-----+-----+-----+
9 3 rows in set (0.00 sec)
```

Listing 16: Default Indexes from Enrollment

2.3 Table Clarifications

We inserted 1000+ rows into AlterTable, UserAccount, and Job as shown below. To be specific, UserAccount is an autogenerated data set, while the other two are real data sets.

```
1 mysql> select count(*) from AlterTable;
2 +-----+
3 | count(*) |
4 +-----+
5 |    45708 |
6 +-----+
7 1 row in set (0.01 sec)
```

Listing 17: AlterTable Count

```
1 mysql> select count(*) from UserAccount;
2 +-----+
3 | count(*) |
4 +-----+
5 |     3000 |
6 +-----+
7 1 row in set (0.26 sec)
```

Listing 18: UserAccount Count

```
1 mysql> select count(*) from Job;
2 +-----+
3 | count(*) |
4 +-----+
5 |     4818 |
6 +-----+
7 1 row in set (0.01 sec)
```

Listing 19: Job Count

3 | Advanced Queries

3.1 User: TagCode & IdealCom

3.1.1 Detailed SQL query

```

1 (
2     SELECT J.Role, J.Preference, J.AvgExperience, J.Qualification, J.Type, ROUND((J.
3         MinSalary + J.MaxSalary)/2, 2) AS AvgSalary,
4     J.CompanyName, J.Benefits, P.Responsibilities, P.Description
5     FROM Job J
6         JOIN UserAccount U ON U.Idealcom = J.CompanyName
7         JOIN Positions P ON P.Role = J.Role
8     WHERE U.UserId = 5
9 )
10 UNION
11 (
12     SELECT J.Role, J.Preference, J.AvgExperience, J.Qualification, J.Type, ROUND((J.
13         MinSalary + J.MaxSalary)/2, 2) AS AvgSalary,
14     J.CompanyName, J.Benefits, P.Responsibilities, P.Description
15     FROM UserAccount U
16         JOIN SOC S ON U.TagCode = S.SOC_code
17         JOIN AlterTable A ON S.SOC_code = A.SOCCode
18         JOIN Job J ON A.AlterTitle = J.Role
19         JOIN Positions P ON P.Role = J.Role
20     WHERE UserId = 5
21 )
22 ORDER BY AvgSalary DESC;
23 );

```

Listing 20: Tree Implementation

3.1.2 Functionality

This advanced query uses TagCode and IdealCom in UserAccount for sorting the list of jobs as an output. We would calculate the AvgSalary for each job and sort it in descending order. Further, we use Union on the basis of paring two results for a complete and direct view of lists of jobs at the first entrance, continuing with updates of TagCode and IdealCom regarding separate parts of tree explorations explained in 3.2 and 3.4.

Role	Preference	AvgExperience	Qualification	Type	AvgSalary	CompanyName	Benefits	Responsibilities	Description
Digital Mark...	Male	6	M.Tech	Part-Time	96.00	Molson Coors B...	(Life and Disability Insura...	Develop and execute dig...	Digital Marketing Mana...
Quality Cont...	Male	8	BBA	Contract	94.50	Paccar	(Transportation Benefits...	Establish and enforce q...	Quality Control Manag...
Quality Assu...	Both	10	BCA	Part-Time	93.00	CenterPoint Ener...	(Flexible Spending Accou...	Lead the QA team, set q...	Quality Assurance Man...
Customer S...	Female	9	MBA	Full-Time	92.00	Molson Coors B...	(Life and Disability Insura...	Build and maintain stron...	Customer Success Ma...
Quality Cont...	Both	9	B.Com	Contract	90.50	Prudential plc	(Transportation Benefits...	Establish and enforce q...	Quality Control Manag...
Quality Assu...	Female	5	PhD	Temporary	89.50	Balfour Beatty	(Transportation Benefits...	Lead the QA team, set q...	Quality Assurance Man...
Quality Cont...	Male	7	M.Com	Full-Time	89.00	Advance Auto P...	(Life and Disability Insura...	Establish and enforce q...	Quality Control Manag...
User Interfa...	Male	9	MBA	Contract	88.00	Molson Coors B...	(Employee Assistance Pr...	Create visually appealin...	User Interface Designe...
Quality Cont...	Female	9	M.Com	Contract	86.50	Albertsons	(Legal Assistance, Bonus...	Establish and enforce q...	Quality Control Manag...
Quality Assu...	Both	7	M.Com	Full-Time	83.50	Iuka Resources	(Life and Disability Insura...	Lead the QA team, set q...	Quality Assurance Man...
Clinical Nur...	Male	6	M.Com	Full-Time	83.00	Molson Coors B...	(Transportation Benefits...	Specialize in a specific a...	A Clinical Nurse Specia...
Quality Assu...	Female	10	MCA	Part-Time	83.00	Aviva	(Tuition Reimbursement...	Lead the QA team, set q...	Quality Assurance Man...
Quality Assu...	Male	8	MCA	Intern	82.00	Deutsche Bahn...	(Employee Assistance Pr...	Lead the QA team, set q...	Quality Assurance Man...
Quality Assu...	Female	7	B.Com	Full-Time	81.50	American Family...	(Health Insurance, Retire...	Lead the QA team, set q...	Quality Assurance Man...
Quality Cont...	Male	8	BA	Temporary	81.00	China COSCO Sh...	(Health Insurance, Retire...	Establish and enforce q...	Quality Control Manag...

Figure 3: User: TagCode & IdealCom Result

3.2 Finding Suitable Jobs

3.2.1 Detailed SQL query

```

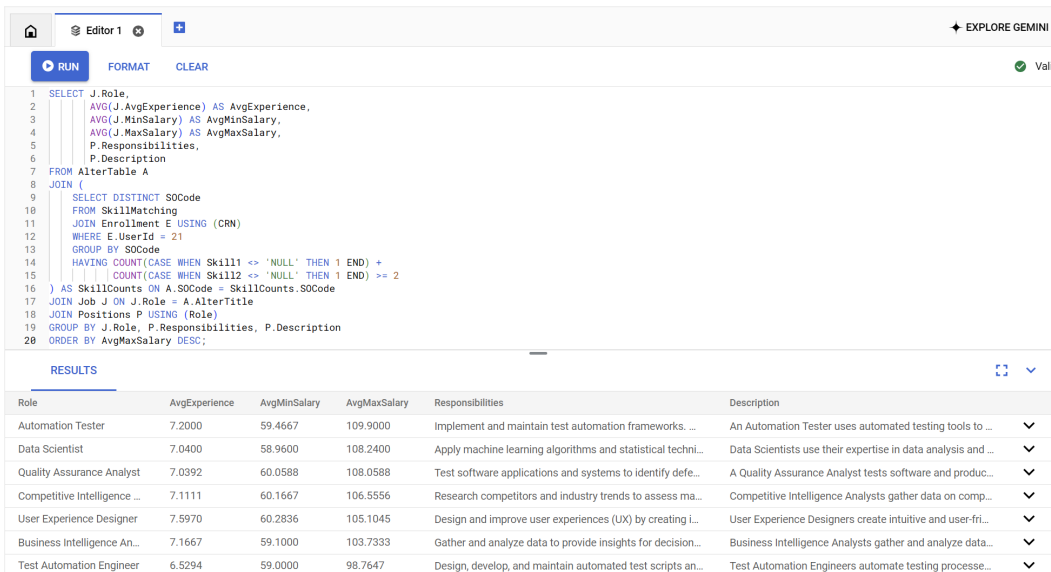
1 SELECT J.Role,
2       AVG(J.AvgExperience) AS AvgExperience,
3       AVG(J.MinSalary) AS AvgMinSalary,
4       AVG(J.MaxSalary) AS AvgMaxSalary,
5       P.Responsibilities,
6       P.Description
7 FROM AlterTable A
8 JOIN (
9     SELECT DISTINCT SOCode
10    FROM SkillMatching
11   JOIN Enrollment E USING (CRN)
12  WHERE E.UserId = 21
13  GROUP BY SOCode
14  HAVING COUNT(CASE WHEN Skill11 <> 'NULL' THEN 1 END) +
15         COUNT(CASE WHEN Skill12 <> 'NULL' THEN 1 END) >= 2
16 ) AS SkillCounts ON A.SOCode = SkillCounts.SOCode
17 JOIN Job J ON J.Role = A.AlterTitle
18 JOIN Positions P USING (Role)
19 GROUP BY J.Role, P.Responsibilities, P.Description
20 ORDER BY AvgMaxSalary DESC;

```

Listing 21: Correlation between Course and Job

3.2.2 Functionality

Based on the enrollment data provided by the user, the system selects suitable job opportunities. The integration with the SkillMatching database allows us to identify relevant skills acquired through courses, which are applicable to future job roles. Consequently, we can select real job positions that require at least two of the skills users have learned. The jobs are grouped by roles, as our goal is to present a concise list of job options rather than delve into detailed job data. Users can explore specific job details later on at their convenience.



Role	AvgExperience	AvgMinSalary	AvgMaxSalary	Responsibilities	Description
Automation Tester	7.2000	59.4667	109.9000	Implement and maintain test automation frameworks. ...	An Automation Tester uses automated testing tools to ...
Data Scientist	7.0400	58.9600	108.2400	Apply machine learning algorithms and statistical techni...	Data Scientists use their expertise in data analysis and ...
Quality Assurance Analyst	7.0392	60.0588	108.0588	Test software applications and systems to identify defe...	A Quality Assurance Analyst tests software and produc...
Competitive Intelligence ...	7.1111	60.1667	106.5556	Research competitors and industry trends to assess ma...	Competitive Intelligence Analysts gather data on comp...
User Experience Designer	7.5970	60.2836	105.1045	Design and improve user experiences (UX) by creating i...	User Experience Designers create intuitive and user-fri...
Business Intelligence An...	7.1667	59.1000	103.7333	Gather and analyze data to provide insights for decision...	Business Intelligence Analysts gather and analyze data...
Test Automation Engineer	6.5294	59.0000	98.7647	Design, develop, and maintain automated test scripts an...	Test Automation Engineers automate testing processe...

Figure 4: Finding Suitable Jobs Query Result

3.3 Courses Plan

3.3.1 Detailed SQL query

```

1 SELECT c.CRN, c.CourseName, c.CreditHour, c.Description, k.Skill1, k. Skill2,
2 FROM UserAccount u
3 JOIN SkillMatching k ON u.TagCode = k.S0Code
4 JOIN CourseInfo c USING (CRN)
5 WHERE u.UserId = 990
6 AND c.CRN NOT IN (
7     SELECT CRN
8     FROM Enrollment
9     WHERE UserId = 990
10 );

```

Listing 22: Course Plan Implementation

3.3.2 Functionality

This advanced query is designed to generate a personalized future course plan for a specific user by leveraging their account and enrollment data. The process is as follows:

1. **User Identification:** The query begins by referencing the user's selected **TagCode** in the **UserAccount** table to identify their associated skill set.
2. **Skill Matching:** Using the **SkillMatching** table, the query retrieves all relevant courses that align with the user's skills.
3. **Course Filtering:** It then filters out any courses that the user has already enrolled in or is currently taking, as recorded in the **Enrollment** table. This ensures that the user is presented with new and relevant course options.

The resulting course plan is tailored to the user's academic and career objectives, providing a streamlined list of courses that they have not yet taken but are pertinent to their skill set. The figure below illustrates a possible course plan for a particular user. The output is limited to fewer than 15 rows, as the enrollment data for the user is auto-generated and the **SkillMatching** table is relatively small.

```

1 SELECT c.CRN, c.CourseName, c.CreditHour, c.Description, k.Skill1, k. Skill2
2 FROM UserAccount u
3 JOIN SkillMatching k ON u.TagCode = k.S0Code
4 JOIN CourseInfo c USING (CRN)
5 WHERE u.UserId = 990
6 AND c.CRN NOT IN (
7     SELECT CRN
8     FROM Enrollment
9     WHERE UserId = 990
10 );

```

CRN	CourseName	CreditHour	Description	Skill1	Skill2
ECE385	Digital Systems Laboratory	3	Design, build, and test digital systems using transistor-transistor logic (...)	Verilog	Hardware Design
ECE391	Computer Systems Engineering	4	Concepts and abstractions central to the development of modern com...	Operating Systems	Ubuntu
ECE408	Applied Parallel Programming	4	Parallel programming with emphasis on developing applications for pr...	CUDA	High Performance Compting
ECE411	Computer Organization & Design	4	Basic computer organization and design: integer and floating-point co...	Computer Design	Computer Architecture
ECE424	Computer Security II	3	Program security, trusted base, privacy, anonymity, non-interference, inf...	Communication Security	NULL
ECE448	Artificial Intelligence	3	Major topics in and directions of research in artificial intelligence: basic...	Python	Gen AI

Figure 5: Finding Course Plan Result

3.3.3 Future Enhancements: Stage 4

In Stage 4, we aim to enhance the course planning procedure by incorporating prerequisites for each course. This will ensure that all necessary foundational courses are accounted for, providing a more comprehensive and logical course progression for the user. Consequently, the output list is expected to be more extensive and detailed.

CRN	CourseName	CreditHour	Description	
CS173	Discrete Structures	3	Discrete mathematical structures frequently encountered in the study of Computer ...	✓
CS225	Data Structures	4	Data abstractions: elementary data structures (lists, stacks, queues, and trees) and t...	✓
ECE110	Introduction to Electronics	3	Introduction to selected fundamental concepts and principles in electrical engineeri...	✓
ECE220	Computer Systems & Programming	4	Advanced use of LC-3 assembly language for I/O and function calling convention. C ...	✓
ECE313	Probability with Engrg Applic	3	Probability theory with applications to engineering problems such as the reliability o...	✓
ECE385	Digital Systems Laboratory	3	Design, build, and test digital systems using transistor-transistor logic (TTL), System...	✓
ECE391	Computer Systems Engineering	4	Concepts and abstractions central to the development of modern computing syste...	✓
ECE408	Applied Parallel Programming	4	Parallel programming with emphasis on developing applications for processors with...	✓
ECE411	Computer Organization & Design	4	Basic computer organization and design: integer and floating-point computer arithm...	✓
ECE424	Computer Security II	3	Program security, trusted base, privacy, anonymity, non-interference, information flo...	✓

Figure 6: Ideal Course Plan Result - Table 1: Course Information

CRN	CourseName	CreditHour	Description	PreID	Pre1	Pre2	
ECE385	Digital Systems Laboratory	3	Design, build, and test digital systems using transistor-transistor l...	1	ECE110	ECE220	✓
ECE391	Computer Systems Engineeri...	4	Concepts and abstractions central to the development of modern ...	1	ECE220		✓
ECE391	Computer Systems Engineeri...	4	Concepts and abstractions central to the development of modern ...	2	CS233		✓
ECE408	Applied Parallel Programming	4	Parallel programming with emphasis on developing applications f...	1	ECE220		✓
ECE411	Computer Organization & De...	4	Basic computer organization and design: integer and floating-poi...	1	ECE385	ECE391	✓
ECE411	Computer Organization & De...	4	Basic computer organization and design: integer and floating-poi...	2	ECE385	CS341	✓
ECE424	Computer Security II	3	Program security, trusted base, privacy, anonymity, non-interferen...	1	CS461		✓
ECE448	Artificial Intelligence	3	Major topics in and directions of research in artificial intelligence: ...	1	CS225	CS361	✓
ECE448	Artificial Intelligence	3	Major topics in and directions of research in artificial intelligence: ...	2	CS225	STAT361	✓
ECE448	Artificial Intelligence	3	Major topics in and directions of research in artificial intelligence: ...	3	CS225	ECE313	✓
ECE448	Artificial Intelligence	3	Major topics in and directions of research in artificial intelligence: ...	4	CS225	MATH362	✓
ECE448	Artificial Intelligence	3	Major topics in and directions of research in artificial intelligence: ...	5	CS225	MATH461	✓
ECE448	Artificial Intelligence	3	Major topics in and directions of research in artificial intelligence: ...	6	CS225	MATH463	✓
ECE448	Artificial Intelligence	3	Major topics in and directions of research in artificial intelligence: ...	7	CS225	STAT400	✓
ECE448	Artificial Intelligence	3	Major topics in and directions of research in artificial intelligence: ...	8	CS225	BIOE310	✓

Figure 7: Ideal Course Plan Result - Table 2: Prerequisites

3.4 Recommending relevant companies

3.4.1 Detailed SQL query

```

1 SELECT C.Name, C.Size, C.CEO, C.Industry, C.Website, ROUND(AVG(J.MinSalary), 2) AS
   MinAvgSalary, ROUND(AVG(J.MaxSalary), 2) AS MaxAvgSalary
2 FROM Company C
3     JOIN Job J ON C.Name = J.CompanyName
4 WHERE C.Industry = (
5     SELECT Industry
6     FROM Company
7     JOIN UserAccount
8         ON Company.Name = UserAccount.Idealcom
9     WHERE UserAccount.UserId = 5
10 )
11 GROUP BY C.Name, C.Size, C.CEO, C.Industry, C.Website
12 ORDER BY MaxAvgSalary DESC, MinAvgSalary DESC;

```

Listing 23: Salary and Role Analysis

3.4.2 Functionality

In addition to recommending jobs at their desired company, we believe that users should have access to information about the company. Therefore, we have designed an extensive “recommendation” system that shows the user information about all companies in the same industry as his/her favorite company. On one hand, it allows users to have more practical knowledge about their chosen industry, and on the other hand, it allows them to learn about and consider other companies in their field, thus expanding their employment opportunities.

Name	Size	CEO	Industry	Website	MinAvgSalary	MaxAvgSalary
Mahindra & Mahindra	76137	Anish Shah	Automotive	www.mahindra.com	62.40	115.60
Fiat Chrysler Automobiles N.V.	65070	Carlos Tavares	Automotive	https://www.fcagroup.com/	61.67	108.83
BMW Group	61352	Oliver Zipse	Automotive	www.bmwgroup.com	61.17	108.50
Geely Automobile Holdings	73680	Daniel Li Donghui	Automotive	https://www.geely.com/	60.57	108.43
BYD Company	116761	Wang Chuanfu	Automotive	https://www.byd.com/	62.00	108.00
Maruti Suzuki India	89385	Kenichi Ayukawa	Automotive	www.marutisuzuki.com	58.25	107.75
Hero MotoCorp	64145	Pawan Munjal	Automotive	https://www.heromotocorp.com/	58.25	107.50
Bajaj Auto	83172	Rakesh Sharma	Automotive	https://www.bajajauto.com/	61.00	106.57
Great Wall Motors	89065	Wei Jianjun	Automotive	http://www.gwm-global.com/	59.33	105.89
Continental AG	84224	Nikolai Setzer	Automotive	www.continental-corporation.com	59.00	105.00
Daimler AG (now known as Mercedes-Benz AG)	70638	Ola K Ilenius	Automotive	www.daimler.com	60.50	104.17
NIO Inc.	65693	William Li Bin	Automotive	https://www.nio.com/	59.50	104.00
Tata Motors	80183	Guenter Butschek	Automotive	https://www.tatamotors.com/	60.00	103.50
Ford Motor Company	56876	Jim Farley	Automotive	https://www.ford.com/	61.40	102.70
Volkswagen Group	73898	Herbert Diess	Automotive	www.volkswagenag.com	60.57	101.29

Figure 8: Recommending Relevant Companies Query Result

4 | Indexing Analysis

4.1 User: TagCode & IdealCom

4.1.1 Initial State: Before Indexing

Before applying any additional indexes, the database utilized primary keys and foreign keys across five joined tables within the query. To evaluate the query performance, the **EXPLAIN ANALYZE** command was executed. The analysis revealed an overall cost of 3688.89, primarily attributed to the nested loop and union operations.

```
| -> Sort: AvgSalary DESC (cost=3688.89..3688.89 rows=2156) (actual time=1.053..1.058 rows=27 loops=1)
  -> Table scan on <union temporary> (cost=1056.11..1085.55 rows=2156) (actual time=0.904..0.911 rows=27 loops=1)
    -> Union materialize with deduplication (cost=1056.10..1056.10 rows=2156) (actual time=0.899..0.899 rows=27 loops=1)
      -> Nested loop inner join (cost=2.80 rows=4) (actual time=0.074..0.102 rows=4 loops=1)
        -> Filter: (J.'Role' is not null) (cost=1.40 rows=4) (actual time=0.035..0.039 rows=4 loops=1)
          -> Index lookup on J using CompanyName (CompanyName='Molson Coors Beverage') (cost=1.40 rows=4) (actual time=0.034..0.037 rows=4 loops=1)
          -> Single-row index lookup on P using PRIMARY (Role=J.'Role') (cost=0.28 rows=1) (actual time=0.015..0.015 rows=1 loops=4)
        -> Nested loop inner join (cost=837.68 rows=2152) (actual time=0.519..0.663 rows=23 loops=1)
          -> Nested loop inner join (cost=84.44 rows=108) (actual time=0.413..0.481 rows=2 loops=1)
            -> Filter: (A.SOCODE = '11-3051') (cost=46.64 rows=108) (actual time=0.099..0.147 rows=108 loops=1)
              -> Covering index lookup on A using SOCODE (SOCODE='11-3051') (cost=46.64 rows=108) (actual time=0.097..0.124 rows=108 loops=1)
              -> Single-row index lookup on P using PRIMARY (Role=A.AlterTitle) (cost=0.25 rows=1) (actual time=0.003..0.003 rows=0 loops=108)
            -> Index lookup on J using Role (Role=A.AlterTitle) (cost=5.00 rows=20) (actual time=0.086..0.089 rows=12 loops=2)
```

Figure 9: Query Cost Before Adding Additional Indexes

4.1.2 Tradeoffs

Three indexing strategies were explored to optimize query performance:

1. **Strategy 1:** Add indexes on Job.MinSalary and Job.MaxSalary.
2. **Strategy 2:** Add an index on Position.Responsibilities.
3. **Strategy 3:** Add an index on SOC.Name.

Despite implementing these strategies, the performance metrics remained largely unchanged. The query costs persisted at similar levels across all attempts, indicating that these additional indexes did not provide the expected optimization benefits.

```
| -> Sort: AvgSalary DESC (cost=3688.89..3688.89 rows=2156) (actual time=0.745..0.750 rows=27 loops=1)
  -> Table scan on <union temporary> (cost=1056.11..1085.55 rows=2156) (actual time=0.704..0.712 rows=27 loops=1)
    -> Union materialize with deduplication (cost=1056.10..1056.10 rows=2156) (actual time=0.700..0.700 rows=27 loops=1)
      -> Nested loop inner join (cost=2.80 rows=4) (actual time=0.048..0.070 rows=4 loops=1)
        -> Filter: (J.'Role' is not null) (cost=1.40 rows=4) (actual time=0.035..0.038 rows=4 loops=1)
          -> Index lookup on J using CompanyName (CompanyName='Molson Coors Beverage') (cost=1.40 rows=4) (actual time=0.033..0.036 rows=4 loops=1)
          -> Single-row index lookup on P using PRIMARY (Role=J.'Role') (cost=0.28 rows=1) (actual time=0.008..0.008 rows=1 loops=4)
        -> Nested loop inner join (cost=837.68 rows=2152) (actual time=0.351..0.505 rows=23 loops=1)
          -> Nested loop inner join (cost=84.44 rows=108) (actual time=0.299..0.384 rows=2 loops=1)
            -> Filter: (A.SOCODE = '11-3051') (cost=46.64 rows=108) (actual time=0.042..0.090 rows=108 loops=1)
              -> Covering index lookup on A using SOCODE (SOCODE='11-3051') (cost=46.64 rows=108) (actual time=0.040..0.066 rows=108 loops=1)
              -> Single-row index lookup on P using PRIMARY (Role=A.AlterTitle) (cost=0.25 rows=1) (actual time=0.003..0.003 rows=0 loops=108)
            -> Index lookup on J using Role (Role=A.AlterTitle) (cost=5.00 rows=20) (actual time=0.056..0.059 rows=12 loops=2)
```

Figure 10: Query Cost After Implementing Strategy 1

```

| -> Sort: AvgSalary DESC (cost=3688.89..3688.89 rows=2156) (actual time=1.031..1.036 rows=27 loops=1)
    -> Table scan on <union temporary> (cost=1056.11..1085.55 rows=2156) (actual time=0.985..0.993 rows=27 loops=1)
        -> Union materialize with deduplication (cost=1056.10..1056.10 rows=2156) (actual time=0.979..0.979 rows=27 loops=1)
            -> Nested loop inner join (cost=2.80 rows=4) (actual time=0.069..0.091 rows=4 loops=1)
                -> Filter: (J.'Role' is not null) (cost=1.40 rows=4) (actual time=0.054..0.057 rows=4 loops=1)
                    -> Index lookup on J using CompanyName (CompanyName='Molson Coors Beverage') (cost=1.40 rows=4) (actual time=0.053..0.056 rows=4 loops=1)
                -> Single-row index lookup on P using PRIMARY (Role=J.'Role') (cost=0.28 rows=1) (actual time=0.008..0.008 rows=1 loops=4)
            -> Nested loop inner join (cost=837.68 rows=2152) (actual time=0.491..0.752 rows=23 loops=1)
                -> Nested loop inner join (cost=84.44 rows=108) (actual time=0.300..0.400 rows=2 loops=1)
                    -> Filter: (A.SOCODE = '11-3051') (cost=46.64 rows=108) (actual time=0.043..0.095 rows=108 loops=1)
                        -> Covering index lookup on A using SOCODE (SOCODE='11-3051') (cost=46.64 rows=108) (actual time=0.041..0.070 rows=108 loops=1)
                    -> Single-row index lookup on P using PRIMARY (Role=A.AlterTitle) (cost=0.25 rows=1) (actual time=0.003..0.003 rows=0 loops=108)
                -> Index lookup on J using Role (Role=A.AlterTitle) (cost=5.00 rows=20) (actual time=0.170..0.175 rows=12 loops=2)

```

Figure 11: Query Cost After Implementing Strategy 2

```

| -> Sort: AvgSalary DESC (cost=3688.89..3688.89 rows=2156) (actual time=0.809..0.814 rows=27 loops=1)
    -> Table scan on <union temporary> (cost=1056.11..1085.55 rows=2156) (actual time=0.773..0.781 rows=27 loops=1)
        -> Union materialize with deduplication (cost=1056.10..1056.10 rows=2156) (actual time=0.769..0.769 rows=27 loops=1)
            -> Nested loop inner join (cost=2.80 rows=4) (actual time=0.054..0.078 rows=4 loops=1)
                -> Filter: (J.'Role' is not null) (cost=1.40 rows=4) (actual time=0.039..0.042 rows=4 loops=1)
                    -> Index lookup on J using CompanyName (CompanyName='Molson Coors Beverage') (cost=1.40 rows=4) (actual time=0.038..0.041 rows=4 loops=1)
                -> Single-row index lookup on P using PRIMARY (Role=J.'Role') (cost=0.28 rows=1) (actual time=0.008..0.008 rows=1 loops=4)
            -> Nested loop inner join (cost=837.68 rows=2152) (actual time=0.427..0.564 rows=23 loops=1)
                -> Nested loop inner join (cost=84.44 rows=108) (actual time=0.369..0.437 rows=2 loops=1)
                    -> Filter: (A.SOCODE = '11-3051') (cost=46.64 rows=108) (actual time=0.040..0.092 rows=108 loops=1)
                        -> Covering index lookup on A using SOCODE (SOCODE='11-3051') (cost=46.64 rows=108) (actual time=0.039..0.067 rows=108 loops=1)
                    -> Single-row index lookup on P using PRIMARY (Role=A.AlterTitle) (cost=0.25 rows=1) (actual time=0.003..0.003 rows=0 loops=108)
                -> Index lookup on J using Role (Role=A.AlterTitle) (cost=5.00 rows=20) (actual time=0.059..0.062 rows=12 loops=2)

```

Figure 12: Query Cost After Implementing Strategy 3

4.1.3 Final Call

Based on the analysis, the original indexing strategy—utilizing only primary and foreign keys—proved to be sufficient for optimizing the first advanced query. The addition of indexes on other attributes did not yield any significant performance improvements in terms of query cost. Therefore, it is recommended to retain the original indexing approach without incorporating the additional indexes explored in this study.

4.2 Finding Suitable Jobs

4.2.1 Original Indices

Original indexes are primary keys and foreign keys of 5 joined tables in query. After running explain analyze command, the result shows as below. The overall cost of nested loop inner loop is 1850.53.

```
| -- Sort: AvgMaxSalary DESC (actual time=24.075..24.076 rows=7 loops=1)
| --> Table scan on <temporary> (actual time=23.042..23.049 rows=7 loops=1)
| --> Aggregate using temporary table (actual time=23.039..23.039 rows=7 loops=1)
| --> Nested loop inner join (cost=1850.53 rows=0) (actual time=15.330..20.321 rows=238 loops=1)
| --> Filter: (SkillCounts.SOCODE = A.SOCODE) (cost=223.31 rows=0) (actual time=14.039..14.093 rows=7 loops=1)
| --> Inner hash join (chash)(SkillCounts.SOCODE)=chash(A.SOCODE) (cost=223.31 rows=0) (actual time=14.036..14.077 rows=7 loops=1)
| --> Table scan on SkillCounts (cost=2.50..2.50 rows=0) (actual time=0.394..0.401 rows=3 loops=1)
| --> Materialize (cost=0.00..0.00 rows=0) (actual time=0.392..0.392 rows=3 loops=1)
| --> Filter: ((count((case when (SkillMatching.Skill1 <> 'NULL') then 1 end)) + count((case when (SkillMatching.Skill2 <> 'NULL') then 1 end))) >= 2) (actual time=0.343..0.344 rows=3 loops=1)
| --> Table scan on <temporary> (actual time=0.335..0.336 rows=4 loops=1)
| --> Aggregate using temporary table (actual time=0.333..0.333 rows=4 loops=1)
| --> Nested loop inner join (cost=2.71 rows=13) (actual time=0.195..0.225 rows=5 loops=1)
| --> Covering index lookup on E using PRIMARY (UserId=21) (cost=0.68 rows=3) (actual time=0.072..0.076 rows=3 loops=1)
| --> Index lookup on SkillMatching using PRIMARY (CRN=E.CRN) (cost=0.39 rows=4) (actual time=0.045..0.048 rows=2 loops=3)
| -- Hash
| --> Nested loop inner join (cost=203.80 rows=233) (actual time=1.669..10.882 rows=233 loops=1)
| --> Table scan on P (cost=25.55 rows=233) (actual time=0.551..1.349 rows=233 loops=1)
| --> Covering index lookup on A using PRIMARY (AlterTitle=P.Role) (cost=0.67 rows=1) (actual time=0.041..0.042 rows=1 loops=233)
| --> Index lookup on J using Role (Role=P.Role) (cost=5.83 rows=20) (actual time=0.504..0.885 rows=34 loops=7)
```

Figure 13: Cost Before Indexing

4.2.2 Tradeoffs

- Strategy 1: add indices on Job (MinSalary) and Job (MaxSalary)

```
| -- Sort: AvgMaxSalary DESC (actual time=7.034..7.036 rows=7 loops=1)
| --> Table scan on <temporary> (actual time=7.002..7.004 rows=7 loops=1)
| --> Aggregate using temporary table (actual time=7.000..7.000 rows=7 loops=1)
| --> Nested loop inner join (cost=1616.21 rows=0) (actual time=3.013..4.702 rows=238 loops=1)
| --> Filter: (SkillCounts.SOCODE = A.SOCODE) (cost=223.31 rows=0) (actual time=3.702..3.723 rows=7 loops=1)
| --> Inner hash join (chash)(SkillCounts.SOCODE)=chash(A.SOCODE) (cost=223.31 rows=0) (actual time=3.700..3.717 rows=7 loops=1)
| --> Table scan on SkillCounts (cost=2.50..2.50 rows=0) (actual time=0.126..0.130 rows=3 loops=1)
| --> Materialize (cost=0.00..0.00 rows=0) (actual time=0.125..0.125 rows=3 loops=1)
| --> Filter: ((count((case when (SkillMatching.Skill1 <> 'NULL') then 1 end)) + count((case when (SkillMatching.Skill2 <> 'NULL') then 1 end))) >= 2) (actual time=0.104..0.105 rows=3 loops=1)
| --> Table scan on <temporary> (actual time=0.098..0.099 rows=4 loops=1)
| --> Aggregate using temporary table (actual time=0.097..0.097 rows=4 loops=1)
| --> Nested loop inner join (cost=2.71 rows=13) (actual time=0.057..0.069 rows=5 loops=1)
| --> Covering index lookup on E using PRIMARY (UserId=21) (cost=0.68 rows=3) (actual time=0.041..0.042 rows=3 loops=1)
| --> Index lookup on SkillMatching using PRIMARY (CRN=E.CRN) (cost=0.39 rows=4) (actual time=0.008..0.008 rows=2 loops=3)
| -- Hash
| --> Nested loop inner join (cost=203.80 rows=233) (actual time=0.569..3.081 rows=233 loops=1)
| --> Table scan on P (cost=25.55 rows=233) (actual time=0.533..0.792 rows=233 loops=1)
| --> Covering index lookup on A using PRIMARY (AlterTitle=P.Role) (cost=0.67 rows=1) (actual time=0.009..0.010 rows=1 loops=233)
| --> Index lookup on J using Role (Role=P.Role) (cost=4.99 rows=20) (actual time=0.069..0.137 rows=34 loops=7)
```

Figure 14: Cost of Strategy 1

- Strategy 2: add indices on SkillMatching (Skill1) and SkillMatching (Skill2)

```
| -- Sort: AvgMaxSalary DESC (actual time=27.392..27.394 rows=7 loops=1)
| --> Table scan on <temporary> (actual time=26.130..26.137 rows=7 loops=1)
| --> Aggregate using temporary table (actual time=26.127..26.127 rows=7 loops=1)
| --> Nested loop inner join (cost=1694.32 rows=0) (actual time=15.855..21.664 rows=238 loops=1)
| --> Filter: (SkillCounts.SOCODE = A.SOCODE) (cost=223.31 rows=0) (actual time=14.256..14.309 rows=7 loops=1)
| --> Inner hash join (chash)(SkillCounts.SOCODE)=chash(A.SOCODE) (cost=223.31 rows=0) (actual time=14.254..14.291 rows=7 loops=1)
| --> Table scan on SkillCounts (cost=2.50..2.50 rows=0) (actual time=0.198..0.205 rows=3 loops=1)
| --> Materialize (cost=0.00..0.00 rows=0) (actual time=0.197..0.197 rows=3 loops=1)
| --> Filter: ((count((case when (SkillMatching.Skill1 <> 'NULL') then 1 end)) + count((case when (SkillMatching.Skill2 <> 'NULL') then 1 end))) >= 2) (actual time=0.169..0.171 rows=3 loops=1)
| --> Table scan on <temporary> (actual time=0.163..0.164 rows=4 loops=1)
| --> Aggregate using temporary table (actual time=0.162..0.162 rows=4 loops=1)
| --> Nested loop inner join (cost=2.71 rows=13) (actual time=0.085..0.126 rows=5 loops=1)
| --> Covering index lookup on E using PRIMARY (UserId=21) (cost=0.68 rows=3) (actual time=0.047..0.049 rows=3 loops=1)
| --> Index lookup on SkillMatching using PRIMARY (CRN=E.CRN) (cost=0.39 rows=4) (actual time=0.023..0.025 rows=2 loops=3)
| -- Hash
| --> Nested loop inner join (cost=203.80 rows=233) (actual time=0.676..13.457 rows=233 loops=1)
| --> Table scan on P (cost=25.55 rows=233) (actual time=0.551..1.349 rows=233 loops=1)
| --> Covering index lookup on A using PRIMARY (AlterTitle=P.Role) (cost=0.67 rows=1) (actual time=0.050..0.052 rows=1 loops=233)
| --> Index lookup on J using Role (Role=P.Role) (cost=5.27 rows=20) (actual time=0.598..1.045 rows=34 loops=7)
```

Figure 15: Cost of Strategy 2

- Strategy 3: add indices on Position (Responsibilities)

```

1 -> Sort: AvgMaxSalary DESC (actual time=8.935..8.936 rows=7 loops=1)
   -> Table scan on <temporary> (actual time=8.904..8.907 rows=7 loops=1)
       -> Aggregate using temporary table (actual time=8.901..8.901 rows=7 loops=1)
           -> Nested loop inner join (cost=1694.32 rows=0) (actual time=3.939..6.391 rows=238 loops=1)
               -> Filter: (SkillCounts.SOCODE = A.SOCODE) (cost=223.31 rows=0) (actual time=3.786..3.818 rows=7 loops=1)
                   -> Inner hash join (<hash>(SkillCounts.SOCODE)=<hash>(A.SOCODE)) (cost=223.31 rows=0) (actual time=3.785..3.810 rows=7 loops=1)
                       -> Table scan on SkillCounts (cost=2.50..2.50 rows=0) (actual time=0.216..0.221 rows=3 loops=1)
                           -> Materialize (cost=0.00..0.00 rows=0) (actual time=0.214..0.214 rows=3 loops=1)
                               -> Filter: ((count((case when (SkillMatching.Skill11 <> 'NULL') then 1 end)) + count((case when (SkillMatching.Skill12 <> 'NULL') then 1 end))) >= 2) (actual time=0.188..0.190 rows=3 loops=1)
                                   -> Table scan on <temporary> (actual time=0.181..0.182 rows=4 loops=1)
                                       -> Aggregate using temporary table (actual time=0.178..0.178 rows=4 loops=1)
                                           -> Nested loop inner join (cost=2.71 rows=13) (actual time=0.052..0.100 rows=5 loops=1)
                                               -> Covering index lookup on E using PRIMARY (UserId=21) (cost=0.68 rows=3) (actual time=0.030..0.037 rows=3 loops=1)
                                                   -> Index lookup on SkillMatching using PRIMARY (CRN=E.CRN) (cost=0.39 rows=4) (actual time=0.015..0.020 rows=2 loops=3)
                                                       -> Hash
                                                           -> Nested loop inner join (cost=203.80 rows=233) (actual time=0.095..3.383 rows=233 loops=1)
                                                               -> Table scan on P (cost=25.55 rows=233) (actual time=0.068..0.232 rows=233 loops=1)
                                                                   -> Covering index lookup on A using PRIMARY (AlterTitle=P.Role) (cost=0.67 rows=1) (actual time=0.012..0.013 rows=1 loops=233)
                                                                       -> Index lookup on J using Role (Role=P.Role) (cost=5.27 rows=20) (actual time=0.304..0.364 rows=34 loops=7)

```

Figure 16: Cost of Strategy 3

- Strategy 4: Combination of Strategy 1 and Strategy 2

```

1 -> Sort: AvgMaxSalary DESC (actual time=6.083..6.084 rows=7 loops=1)
   -> Table scan on <temporary> (actual time=6.039..6.041 rows=7 loops=1)
       -> Aggregate using temporary table (actual time=6.035..6.035 rows=7 loops=1)
           -> Nested loop inner join (cost=1655.27 rows=0) (actual time=3.055..3.824 rows=238 loops=1)
               -> Filter: (SkillCounts.SOCODE = A.SOCODE) (cost=223.31 rows=0) (actual time=2.927..2.946 rows=7 loops=1)
                   -> Inner hash join (<hash>(SkillCounts.SOCODE)=<hash>(A.SOCODE)) (cost=223.31 rows=0) (actual time=2.925..2.942 rows=7 loops=1)
                       -> Table scan on SkillCounts (cost=2.50..2.50 rows=0) (actual time=0.203..0.207 rows=3 loops=1)
                           -> Materialize (cost=0.00..0.00 rows=0) (actual time=0.202..0.202 rows=3 loops=1)
                               -> Filter: ((count((case when (SkillMatching.Skill11 <> 'NULL') then 1 end)) + count((case when (SkillMatching.Skill12 <> 'NULL') then 1 end))) >= 2) (actual time=0.176..0.178 rows=3 loops=1)
                                   -> Table scan on <temporary> (actual time=0.169..0.170 rows=4 loops=1)
                                       -> Aggregate using temporary table (actual time=0.168..0.168 rows=4 loops=1)
                                           -> Nested loop inner join (cost=2.71 rows=13) (actual time=0.067..0.133 rows=5 loops=1)
                                               -> Covering index lookup on E using PRIMARY (UserId=21) (cost=0.68 rows=3) (actual time=0.047..0.048 rows=3 loops=1)
                                                   -> Index lookup on SkillMatching using PRIMARY (CRN=E.CRN) (cost=0.39 rows=4) (actual time=0.024..0.025 rows=2 loops=3)
                                                       -> Hash
                                                           -> Nested loop inner join (cost=203.80 rows=233) (actual time=0.532..2.484 rows=233 loops=1)
                                                               -> Table scan on P (cost=25.55 rows=233) (actual time=0.416..0.462 rows=233 loops=1)
                                                                   -> Covering index lookup on A using PRIMARY (AlterTitle=P.Role) (cost=0.67 rows=1) (actual time=0.007..0.008 rows=1 loops=233)
                                                                       -> Index lookup on J using Role (Role=P.Role) (cost=5.13 rows=20) (actual time=0.072..0.122 rows=34 loops=7)

```

Figure 17: Cost of Strategy 4

4.2.3 Final Call

As shown in the table below, we found that although the performance was increased once any query-related index were added, the best design is the first strategy, namely adding indices on Job (MinSalary) and Job (MaxSalary). However, if we add more indices besides these two, the performance will decrease. Therefore, for the final design, I choose to add indices on Job (MinSalary) and Job (MaxSalary).

Table 1: Comparison

Cost Strategy	Nested Loop Inner Join
0	1850.53
1	1616.21
2	1694.32
3	1694.32
4	1655.27

4.3 Future Course Plan

4.3.1 Before Indexing

The original indexes are the primary and foreign keys of the referenced tables. The dataset this query handles is smaller compared to other queries that involve the Job table. Consequently, before indexing, the cost of the nested loop anti-join is measured at 7.63.

```
| -> Nested loop antijoin (cost=7.55 rows=24) (actual time=1.387..1.413 rows=6 loops=1)
|   -> Nested loop inner join (cost=4.35 rows=8) (actual time=0.110..0.147 rows=8 loops=1)
|     -> Index lookup on k using SOCode (SOCode='15-1211') (cost=1.55 rows=8) (actual time=0.086..0.095 rows=8 loops=1)
|     -> Single-row index lookup on c using PRIMARY (CRN=k.CRN) (cost=0.26 rows=1) (actual time=0.006..0.006 rows=1 loops=8)
|     -> Single-row index lookup on <subquery2> using <auto distinct key> (CRN=k.CRN) (actual time=0.158..0.158 rows=0 loops=8)
|     -> Materialize with deduplication (cost=1.35..1.35 rows=3) (actual time=1.249..1.249 rows=2 loops=1)
|       -> Filter: (Enrollment.CRN is not null) (cost=1.05 rows=3) (actual time=0.040..0.043 rows=2 loops=1)
|       -> Covering index lookup on Enrollment using PRIMARY (UserId=990) (cost=1.05 rows=3) (actual time=0.036..0.039 rows=2 loops=1)
|
```

Figure 18: Cost Before Indexing

4.3.2 Tradeoffs

- Strategy 1: add indices on CourseInfo(CourseName)

```
| -> Nested loop antijoin (cost=7.55 rows=24) (actual time=0.137..0.169 rows=6 loops=1)
|   -> Nested loop inner join (cost=4.35 rows=8) (actual time=0.063..0.097 rows=8 loops=1)
|     -> Index lookup on k using SOCode (SOCode='15-1211') (cost=1.55 rows=8) (actual time=0.051..0.056 rows=8 loops=1)
|     -> Single-row index lookup on c using PRIMARY (CRN=k.CRN) (cost=0.26 rows=1) (actual time=0.005..0.005 rows=1 loops=8)
|     -> Single-row index lookup on <subquery2> using <auto distinct key> (CRN=k.CRN) (actual time=0.009..0.009 rows=0 loops=8)
|     -> Materialize with deduplication (cost=1.35..1.35 rows=3) (actual time=0.060..0.060 rows=2 loops=1)
|       -> Filter: (Enrollment.CRN is not null) (cost=1.05 rows=3) (actual time=0.013..0.016 rows=2 loops=1)
|       -> Covering index lookup on Enrollment using PRIMARY (UserId=990) (cost=1.05 rows=3) (actual time=0.012..0.015 rows=2 loops=1)
|
```

Figure 19: Cost of Strategy 1

- Strategy 2: add indices on SkillMatching (Skill1) and SkillMatching (Skill2)

```
| -> Nested loop antijoin (cost=7.55 rows=24) (actual time=0.181..0.215 rows=6 loops=1)
|   -> Nested loop inner join (cost=4.35 rows=8) (actual time=0.125..0.160 rows=8 loops=1)
|     -> Index lookup on k using SOCode (SOCode='15-1211') (cost=1.55 rows=8) (actual time=0.103..0.106 rows=8 loops=1)
|     -> Single-row index lookup on c using PRIMARY (CRN=k.CRN) (cost=0.26 rows=1) (actual time=0.006..0.006 rows=1 loops=8)
|     -> Single-row index lookup on <subquery2> using <auto distinct key> (CRN=k.CRN) (actual time=0.006..0.006 rows=0 loops=8)
|     -> Materialize with deduplication (cost=1.35..1.35 rows=3) (actual time=0.041..0.041 rows=2 loops=1)
|       -> Filter: (Enrollment.CRN is not null) (cost=1.05 rows=3) (actual time=0.021..0.024 rows=2 loops=1)
|       -> Covering index lookup on Enrollment using PRIMARY (UserId=990) (cost=1.05 rows=3) (actual time=0.020..0.023 rows=2 loops=1)
|
```

Figure 20: Cost of Strategy 2

- Strategy 3: add indices on CourseInfo(CreditHour)

```
| -> Nested loop antijoin (cost=7.55 rows=24) (actual time=0.140..0.162 rows=6 loops=1)
|   -> Nested loop inner join (cost=4.35 rows=8) (actual time=0.062..0.085 rows=8 loops=1)
|     -> Index lookup on k using SOCode (SOCode='15-1211') (cost=1.55 rows=8) (actual time=0.052..0.054 rows=8 loops=1)
|     -> Single-row index lookup on c using PRIMARY (CRN=k.CRN) (cost=0.26 rows=1) (actual time=0.003..0.004 rows=1 loops=8)
|     -> Single-row index lookup on <subquery2> using <auto distinct key> (CRN=k.CRN) (actual time=0.009..0.009 rows=0 loops=8)
|     -> Materialize with deduplication (cost=1.35..1.35 rows=3) (actual time=0.067..0.067 rows=2 loops=1)
|       -> Filter: (Enrollment.CRN is not null) (cost=1.05 rows=3) (actual time=0.056..0.058 rows=2 loops=1)
|       -> Covering index lookup on Enrollment using PRIMARY (UserId=990) (cost=1.05 rows=3) (actual time=0.055..0.057 rows=2 loops=1)
|
```

Figure 21: Cost of Strategy 3

None of the three strategies resulted in a significant reduction in cost, maintaining levels similar to the original configuration. This may be due to the attributes used in the WHERE clause and JOIN primarily being primary keys or foreign keys, leaving little room for optimization. Therefore, we ultimately decided to stick with the original primary key and foreign key indexes.

4.4.1 Original Indices

- Strategy 1: add indices on Company(Industry)
- Strategy 2: add indices on Company(Industry) and Company(CEO)
- Strategy 3: add indices on Company(Industry), Company(CEO), Company(Website) and Company(Size)

Figure 24: Cost of Strategy2

Figure 25: Cost of Strategy3

After add indices on Company(CEO), Company(Website) and Company(Size), we can observe that the costs don't have any change. This may because that although these attributes are in GROUP BY clause, the real aggregation is determined by Company(Name) only, as it is the primary key of table Company.

We decide to select the design of adding index on Company(Industry) only. Since further indices don't have effective influence on cost. They will even reduce the actual time of query and increase the complexity of writing operations on database like INSERT, DELETE, UPDATE. Therefore, we choose Strategy 1 eventually.