UniCareers

Conceptual Design Report

Keeron Huang, Libin Wang, Mingyan Gao, Yanzi Li[1]

[1]University of Illinois Urbana-Champaign

Keywords: Job Exploration, College Career, Education

1 | ER/UML Diagram

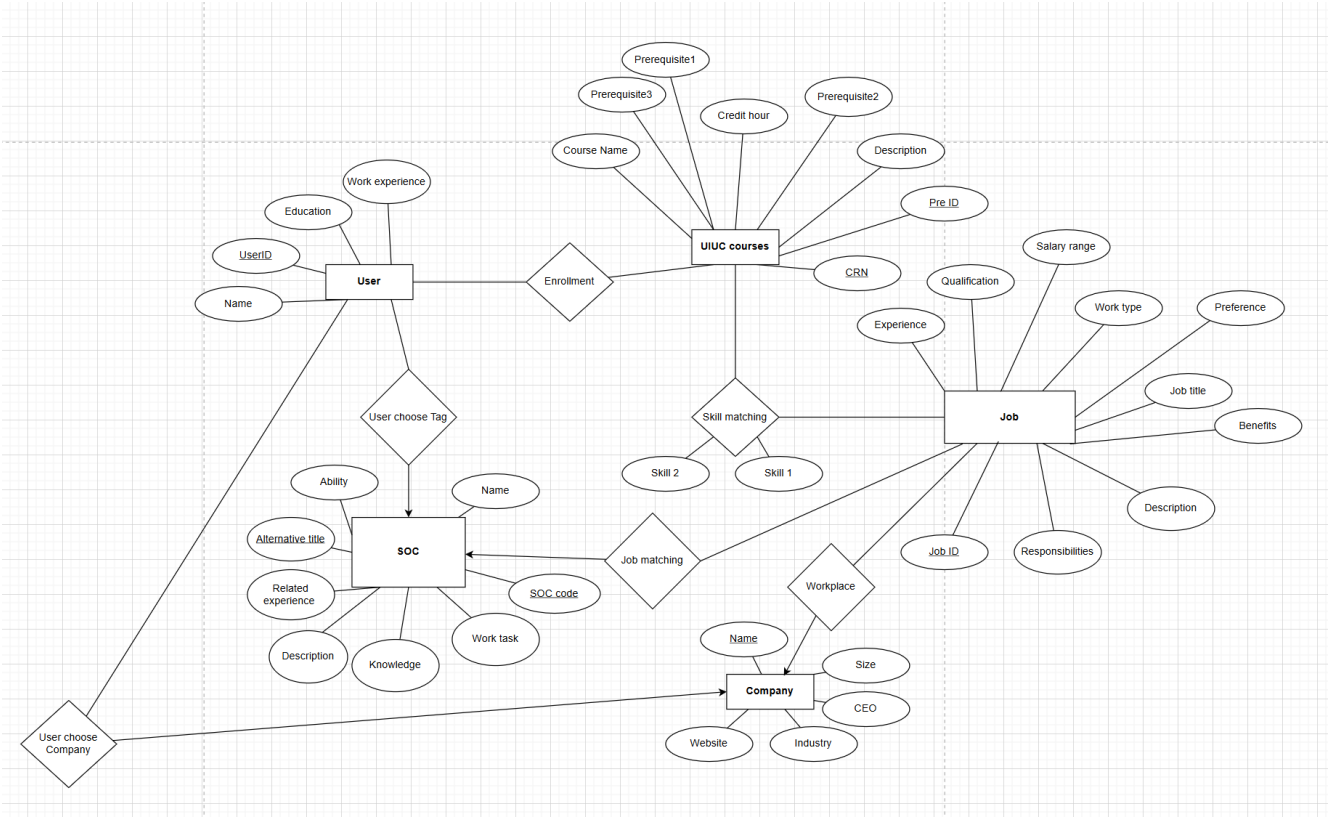

Figure 1: ER Model for Project Database

2 | Entities Descriptions

2.1 User Profile

2.1.1 Attributes

Entity *USER* contains user information including attributes *UserId, Name, Education, Work_Experience*. All of these except *UserId* come from user inputs. *UserId* will be assigned by our database system automatically.

2.1.2 Assumptions

1. User will always input at least its name for registration. Also, for realizing our function, user must select at least one of *SOC Tag* and *Ideal Company* to further search for desired jobs.

2. If user wants to choose ideal company (relation between User and Company) or tag (relation between User and SOC) or completed course (relation between User and UIUC Courses), they must select from corresponding referenced attribute elements.

3. If user inputs his/her work experience, it should be the number of total working hour in the past (including all work types).

4. *Education* should be selected from "B.Com" "B.Tech" "BA" "BBA" "BCA" "M.Com" "M.Tech" "MBA" "MCA" "Phd".
5. *UserId* for a user will not change forever after registration.

2.1.3 Clarification

USER is targeted at user registration and information storage. USER should be an entity because it is an real-world object with more than one non-key attribute and more than one time being the "many" part of "many-to-many" or "many to one" relationships.

2.2 SOC Code

2.2.1 Attributes

Entity *SOC* contains *Name, SOC_Code, Alternative_Title, Related_Experience, Description, Knowledge, Work_Task and Ability* as attributes. They will all come from our datasets – list of SOC occupations. *Name* is the standard name for job while *alternative title* contains all job titles that can substitute corresponding standard name. Attributes like *Ability, Related_Experience, Description, Knowledge, Work_Task* are all descriptive information about specific jobs.

2.2.2 Assumptions

1. SOC Code will not change for job titles.
2. Alternative title will not be null.

2.2.3 Clarification

SOC is mainly for the layering of jobs (as in the graph below) and information about general job types. This entity contains many attributes and shares relationships with others, so it should be an entity other than an attribute.

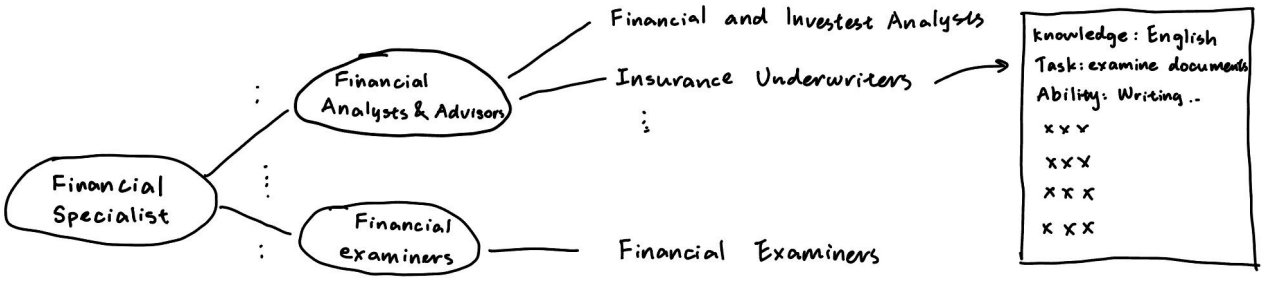

Figure 2: Layering of Jobs

2.3 Job Info

2.3.1 Attributes

Entity *JOB* contains *Experience, Qualification, Salary_Range, Work_Type, Preference, Job_Title, Benefits, Description, Responsibilities, Role, Job_ID*. They will all come from datasets – Kaggle data for previous job recruitment. All these are features of a previous job within a specific company.

2.3.2 Assumptions

1. The company which provided the job would be within the company list.
2. Job title would match the elements of *Alternative Title* in *SOC* Entity.

2.3.3 Clarification

This entity records the information about a specific job within a specific company. It contains numerous attributes to describe the job. Therefore, it should be an entity.

2.4 Company Info

2.4.1 Attributes

Entity *COMPANY* contains attribute *Name, Size, CEO, Industry, Website*. They also from our datasets, which describe the basic information about one company.

2.4.2 Assumptions

1. Company list would only be incremented. Companies that face misfortune (such as bankrupt) will not be removed from the list. Only new companies or companies that revise their information could make change to the table.
2. Attribute *Name* should not be null.

2.4.3 Clarification

This entity includes information for companies. Considering that some users might intend to learn more about the companies, we present core information for them. Therefore, this entity should be an entity because of its more-than-one non-key attributes and complex relationships with other entities.

2.5 UIUC Course

2.5.1 Attributes

Entity *UIUC_COURSES* contains *Course_Name, Credit_Hour, Description, Prerequisite1, Prerequisite2, Prerequisite3, Pre_ID, Course_ID*. Like the Course displayed on course explorer website, we also want to have some details about courses so that users could arrange their future academic plan accordingly. Therefore, we select those attributes. Note that *Pre_ID* is assigned by ourselves to realize the logic of one course with many prerequisite combos.

2.5.2 Assumptions

1. If a course is deleted, the upper-level course should also change its prerequisite.
2. We assume that a course would only have at most 3 prerequisites.
3. We assume that all prerequisites are valid when inserted into the table. It could be null.

2.5.3 Clarification

Courses are absolutely real objects containing many properties. It has attributes that define it and user could have taken some of the courses. Considering the relationship and richness of attributes, it should be an entity.

3 | Relationships Description

3.1 Tag Selection

3.1.1 Cardinality

Many-to-One (Many users → One SOC)
Users can select only one tag representing their preferred career area, but a tag can be selected by many users.

3.1.2 Description

We choose a one-to-one relationship for tag selection due to application constraints. Users can select a tag to accurately focus their job search within their professional area. The SOC source provides a general classification and only indicates a broad career direction. Therefore, a user can have only one tag representing a general field. SOC (Standard Occupational Classification) codes represent these broad job categories. Multiple users may be interested in the same SOC category, which encompasses a group of occupations.

3.2 Company Selection

3.2.1 Cardinality

Many-to-Many (Many users → Many companies)
Users can select multiple companies they are interested in, and a company can be selected by many users.

3.2.2 Description

A user can express interest in several companies, and many users can be interested in the same company. This relationship reflects the flexible and exploratory nature of job searching, where users can explore various opportunities across different companies. Users are not restricted to one company, and companies can attract multiple users based on their preferences and career goals.

3.3 Skill Matching

3.3.1 Cardinality

Many-to-Many (Many UIUC courses → Many jobs)
A course may provide skills useful for many jobs, and a job may require skills taught in various courses.

3.3.2 Description

For the relationship between UIUC courses and jobs, we choose a many-to-many relationship with the relation attribute "Skills." A course may teach skills that are applicable to many different jobs, and a job may require skills that can be acquired through multiple courses. The "Skills" attribute (e.g., Skill1, Skill2, Skill3) represents the overlap between the skills taught in a course and the skills required for a job. This relationship is essential in the Course Matching Job Demo, which provides recommended learning paths for users based on job requirements.

3.4 Job-SOC Matching

3.4.1 Cardinality

Many-to-One (Many jobs → One SOC)
A SOC is a broad classification that can encompass many jobs, but each job belongs to one SOC.

3.4.2 Description

Each job is classified under a single SOC code, but many jobs can be grouped under the same SOC classification. This structure allows the system to organize various job titles within a broader occupational field. Multiple job titles may exist under the same SOC code, representing different roles within the same general career category.

3.5 Workplace Interface

3.5.1 Cardinality

Many-to-One (Many jobs → One company)
A job belongs to one company, and a company can offer many jobs.

3.5.2 Description

A company can offer multiple jobs, but each job is specific to one company. This relationship reflects a constraint within the O*NET database, where each job is linked to a single employer without exceptions. This relationship is used to provide detailed information about available jobs, including the companies offering them.

3.6 Enrollment

3.6.1 Cardinality

Many-to-Many (Many users → Many UIUC courses)
A user can enroll in multiple courses, and each course can be taken by multiple users.

3.6.2 Description

The relationship between UIUC courses and users is many-to-many because students often enroll in a variety of courses, and the same course may be taken by many students. This reflects the flexible academic journey where students take multiple courses to build skills, and courses are open to multiple enrollees. The system can track which users are enrolled in which courses, helping to recommend job opportunities based on the skills acquired from these courses.

4 | Normalization Process

In this project, we use the Boyce-Codd Normal Form (BCNF) to normalize our database schema. BCNF is a more restrictive version of the Third Normal Form (3NF) and ensures that the database is free of anomalies such as redundancy, update issues, and insertion/deletion anomalies. We chose BCNF because of its ability to provide a more simplified and streamlined structure, which aligns well with the specific requirements and complexity of our dataset. In particular, each section of this process focuses on ensuring that our schema adheres to BCNF, resulting in a clean and efficient database design.
The normalization process is divided into several sections:

- **User Profile**: Handles user-specific information such as education, work experience, and preferences.
- **SOC Code**: Manages information on the standard occupational classification system, including alternative titles and associated abilities.
- **Job Info**: Contains data related to job positions, including responsibilities, benefits, and qualifications.
- **Company Info**: Stores details about companies, such as size, CEO, and industry.
- **Course Info**: Holds information on academic courses, including course prerequisites and descriptions.

4.1 User Profile

4.1.1 Key Analysis

The primary key for the `User` table is `userId` (B), which uniquely identifies each user. This ensures that each user is uniquely tracked, even if the users share the same name.
Based on the functional dependencies (FD):

- `userId` (B) $\rightarrow$ `name` (A), `ideal_company` (C), `tag` (D), `education` (E), `experience` (F)

This means that `userId` is a superkey, as it determines all other attributes.

4.1.2 BCNF

In BCNF, for every non-trivial functional dependency $A_1, A_2, ..., A_n \rightarrow B$, $A_1, A_2, ..., A_n$ must be a superkey. In this case, the primary key is `userId`, meaning all non-key attributes must depend solely on this key.

- `name`, `ideal_company`, `tag`, `education`, and `experience` depend directly on the primary key `userId`. There are no partial dependencies, and the key fully determines these attributes, ensuring BCNF compliance.

4.1.3 Corresponding DDL

```
CREATE TABLE User (
    userId BIGINT PRIMARY KEY,
    name VARCHAR(255),
    ideal_company VARCHAR(255),
    tag VARCHAR(255),
    education VARCHAR(255),
    experience INT,
    FOREIGN KEY (tag) REFERENCES SOC(Altertitle),
    FOREIGN KEY (ideal_company) REFERENCES Company(Name)
);
```

4.2 SOC Code

4.2.1 Key Analysis

Based on the functional dependencies (FD):

- `AlterTitle`, `SOC_code` (B, A) $\rightarrow$ `Name`, `Ability`, `Related_Experience`, `Knowledge`, `Work_Task`, `Description` (C, D, E, F, G, H)
- `SOC_code` (A) $\rightarrow$ `Name`, `Ability`, `Related_Experience`, `Knowledge`, `Work_Task`, `Description` (C, D, E, F, G, H)

The combination of `AlterTitle` and `SOC_code` (B, A) is the candidate key and will serve as the primary composite key for this relation. Also, (B, A) is also the candidate key.

4.2.2 BCNF

In BCNF, for every nontrivial functional dependency $X \rightarrow Y$, X must be a superkey. Let's evaluate the FDs:

- `AlterTitle`, `SOC_code` (B, A) $\rightarrow$ `Name`, `Ability`, `Related_Experience`, `Knowledge`, `Work_Task`, `Description` (C, D, E, F, G, H): This dependency satisfies BCNF because the composite key (B, A) is a superkey.
- `SOC_code` (A) $\rightarrow$ `Name`, `Ability`, `Related_Experience`, `Knowledge`, `Work_Task`, `Description` (C, D, E, F, G, H): This violates BCNF because `SOC_code` alone is not a superkey (the superkey is the composite of `AlterTitle` and `SOC_code`).

To resolve this violation of BCNF, we calculate A^+ = ACDEFGH; Therefore, we can decompose the relation into R1(A, C, D, E, F, G, H) and R2(B, A):

- *SOC*(`SOC_code`, `Name`, `Ability`, `Related_Experience`, `Knowledge`, `Work_Task`, `Description`) where `SOC_code` (A) is the key.
- *AlterTable*(`AlterTitle`, `SOC_code`) where `AlterTitle` (B) and `SOC_code` (A) form the composite key.

4.2.3 Corresponding DDL

- **SOC Table**

```
CREATE TABLE SOC (
    SOC_code VARCHAR(255),
    name VARCHAR(255),
    ability LONGTEXT,
    relatedEx LONGTEXT,
    knowledge LONGTEXT,
    worktask LONGTEXT,
    description LONGTEXT,
    PRIMARY KEY (SOC_code)
);
```

- **Alternative Title Table**

```
CREATE TABLE AlterTable (
    AlterTitle VARCHAR(255),
    SOC_code VARCHAR(255),
    PRIMARY KEY (AlterTitle, SOC_code),
    FOREIGN KEY (SOC_code) REFERENCES SOC(SOC_code)
);
```

4.3 Job Info

4.3.1 Key Analysis

Based on the functional dependencies (FD):

- `JobID` (A) $\rightarrow$ `Role` (B), `Responsibilities` (C), `Description` (E), `Benefits` (F), `JobTitle` (G), `Company_Name` (D), `Preference` (H), `WorkType` (I), `Salary_Range` (J), `Qualification` (K), `Experience` (L)
- `Role` (B) $\rightarrow$ `Responsibilities` (C), `Description` (E), `JobTitle` (G)
- `Responsibilities` (C) $\rightarrow$ `Role` (B)
- `Description` (E) $\rightarrow$ `Role` (B)

From these FDs, we can see:

- `JobID` (A) is a superkey, as it determines most other attributes.
- `Role` (B) also plays a crucial role in determining some attributes like `Responsibilities`, `Description`, and `JobTitle`.

4.3.2 BCNF

To satisfy the BCNF condition, we need to ensure that for every nontrivial functional dependency $X \to Y$, X must be a superkey. Let's evaluate the FDs:

- `JobID` (A) $\to$ `Role` (B), `Company_Name` (D), `Benefits` (F), `Preference` (H), `WorkType` (I), `Salary_Range` (J), `Qualification` (K), `Experience` (L): Since `JobID` is a superkey, this satisfies BCNF.
- `Role` (B) $\to$ `Responsibilities` (C), `Description` (E), `JobTitle` (G): Since `Role` is not a superkey, this violates BCNF.
- `Responsibilities` (C) $\to$ `Role` (B): This also violates BCNF.
- `Description` (E) $\to$ `Role` (B): This also violates BCNF.

To resolve these violations, we calculate B^+ = BCEG, Therefore, we decompose the relation into R1(B, C, E, G) and R2(A, B, D, F, H, I, J, K, L):

- *Position*(`Role`, `Responsibilities`, `Description`, `JobTitle`) where `Role` (B) is the key.
- *Job*(`JobID`, `Role`, `Company_Name`, `Benefits`, `Preference`, `WorkType`, `Salary_Range`, `Qualification`, `Experience`) where `JobID` (A) is the key.

4.3.3 Corresponding DDL

- **Position Info**

```
CREATE TABLE Position (
    Role VARCHAR(255),
    Responsibilities  LONGTEXT,
    Description  LONGTEXT,
    JobTitle VARCHAR(255),
    PRIMARY KEY (Role)
);
```

- **Job Info**

```
CREATE TABLE Job(
    JobID VARCHAR(255),
    Role VARCHAR(255),
    Company_Name VARCHAR(255),
    Benefits VARCHAR(255),
    Preference VARCHAR(255),
    WorkType VARCHAR(255),
    Salary_Range VARCHAR(255),
    Qualification VARCHAR(255),
    Experience VARCHAR(255),
    PRIMARY KEY (JobID),
    FOREIGN KEY (Role) REFERENCES AlterTable(AlterTitle)
);
```

4.4 Company Info

4.4.1 Key Analysis

Based on the functional dependencies (FD) shown:

- `Name` $(A) \rightarrow$ `Size`, `CEO`, `Website`, `Industry` (B, C, D, E)

We can infer the following about the candidate keys:

- The attribute `Name` (A) is a superkey. It determines all other attributes of the table.
- Since `Name` uniquely identifies all attributes, it is also the primary key.

4.4.2 BCNF

To satisfy the BCNF condition, we need to ensure that for every nontrivial functional dependency $X \rightarrow Y$, X must be a superkey. Let's evaluate the FDs:

- `Name` $(A) \rightarrow$ `Size`, `CEO`, `Website`, `Industry` (B, C, D, E): Here, `Name` is a superkey, and this dependency satisfies BCNF.

Since `Name` is the only candidate key and determines all other attributes, the `Company` table is already in BCNF. No further decomposition is required.

4.4.3 Corresponding DDL

```
CREATE TABLE Company (
    Name VARCHAR(255),
    Size INT,
    CEO VARCHAR(255),
    Website VARCHAR(255),
    Industry VARCHAR(255),
    PRIMARY KEY (Name)
);
```

4.5 Course Info

4.5.1 Key Analysis

Based on the functional dependencies (FD) shown:

- `CRN` $(A) \rightarrow$ `Description`, `Credit_Hour`, `Course_Name` (C, D, E)
- `Course_Name` $(E) \rightarrow$ `CRN` (A)
- `CRN, PreID` $(A, B) \rightarrow$ `Pre1`, `Pre2`, `Pre3`, `Description`, `Credit_Hour`, `Course_Name` (F, G, H, C, D, E)

The candidate keys are:

- `PreID, CRN` (B, A)
- `PreID, Course_Name` (B, E)

The first candidate key is designed to be primary key.

4.5.2 BCNF

To satisfy the BCNF condition, we need to ensure that for every nontrivial functional dependency $X \to Y$, X must be a superkey. Let's evaluate the FDs:

- `CRN` (A) → `Description`, `Credit_Hour`, `Course_Name` (C, D, E): Since `CRN` is part of the composite key, this satisfies BCNF.
- `Course_Name` (E) → `CRN` (A): This dependency violates BCNF because `Course_Name` is not a superkey.
- `CRN, PreID` (A, B) → `Pre1`, `Pre2`, `Pre3`, `Description`, `Credit_Hour`, `Course_Name` (F, G, H, C, D, E): This dependency satisfies BCNF because the composite key (A, B) is a superkey.

To resolve the violation of BCNF, we calculate A^+ = ACDE. Therefore, we decompose the relation into R1(A, C, D, E) and R2(A, B, F, G, H):

- $CourseInfo$(`CRN`, `Description`, `Credit_Hour`, `Course_Name`) where `CRN` → `Description`, `Credit_Hour`, `Course_Name`.
- $Prerequisite$(`CRN`, `PreID`, `Pre1`, `Pre2`, `Pre3`) where `CRN, PreID` → `Pre1`, `Pre2`, `Pre3`.

4.5.3 Corresponding DDL

```
CREATE TABLE CourseInfo (
    CRN VARCHAR(255),
    Description VARCHAR(255),
    Credit_Hour INT,
    Course_Name VARCHAR(255),
    PRIMARY KEY (CRN)
);


CREATE TABLE Prerequisite (
    CRN VARCHAR(255),
    PreID VARCHAR(255),
    Pre1 VARCHAR(255),
    Pre2 VARCHAR(255),
    Pre3 VARCHAR(255),
    PRIMARY KEY (CRN, PreID),
    FOREIGN KEY (CRN) REFERENCES CourseInfo(CRN)
);
```

5 | Translated Relational Schema

According to the normalization above, we can translate our ER model into the following relational schema.

5.1 User Profile

This table contains data of the profile of the users, such as the name, the work experience.

```
User(
    user_id: BIGINT [PK],
    name: VARCHAR(255),
    idealCom: VARCHAR(255) [FK to Company.Name],
    tag: VARCHAR(255) [FK to SOC.Altertitle],
```

```
    education: VARCHAR(255),
    experience: INT
)
```

5.2 SOC Code

This table contains data of SOC-Code, and all of their corresponding job titles.

```
AlterTable(
    AlterTitle: VARCHAR(255) [PK],
    SOC_code: VARCHAR(255) [PK] [FK to SOC.Code]
)
```

This table contains basic information of each occupation in SOC-Code, for example, knowledge, work task, description etc.

```
SOC(
    SOC_code: VARCHAR(255) [PK],
    name: VARCHAR(255),
    ability: LONGTEXT,
    relatedEx: LONGTEXT,
    knowledge: LONGTEXT,
    worktask: LONGTEXT,
    description: LONGTEXT
)
```

5.3 Job Info

This table contains data of jobs that our group searched from real database, including gender preference, qualification and work type etc.

```
Job(
    JobID: BIGINT [PK],
    Role: VARCHAR(255) [FK to AlterTable.AlterTitle],
    Preference: VARCHAR(255),
    WorkExperience: INT,
    Qualification: VARCHAR(255),
    Type: VARCHAR(255),
    Salary: VARCHAR(255),
    CompanyName: VARCHAR(255),
    Benefits: LONGTEXT
)
```

This table contains basic information relating to the role of the job, which means no matter what company or what gender the employer is, these attributes will always be the same.

```
Position(
    Role: VARCHAR(255) [PK] [FK to Job.Role],
    Responsibilities: LONGTEXT,
    Description: LONGTEXT,
    Jobtitle: VARCHAR(255)
)
```

5.4 Company Info

This table contains detailed information of companies that our group searched from real database, including size, website, etc.

```
Company(
    Name: VARCHAR(255) [PK],
    CEO: VARCHAR(255),
    Size: INT,
    Industry: VARCHAR(255),
    Website: VARCHAR(255)
)
```

5.5 UIUC Course

This table records the prerequisite requirements for courses of ECE and CS majors in UIUC.

```
Prerequisite(
    PreID: BIGINT [PK],
    CRN: VARCHAR(255) [PK] [FK to Courses.CRN],
    Pre1: VARCHAR(255),
    Pre2: VARCHAR(255),
    Pre3: VARCHAR(255)
)
```

This table contains detailed information of courses of ECE and CS majors in UIUC.

```
Courses(
    CRN: VARCHAR(255) [PK],
    Name: VARCHAR(255),
    Credit_Hour: INT,
    Description: LONGTEXT
)
```

5.6 Course-Job matching

This table records the relationship between UIUC courses and jobs we selected. It will record the skill that both course and job require.

```
Skill-Matching(
    Course: VARCHAR(255) [PK] [FK to Courses.CRN],
    Job: BIGINT [PK] [FK to Job.JobID],
    Skill1: VARCHAR(255),
    Skill2: VARCHAR(255)
)
```

5.7 User Enrollment

This table records the course enrollments of users.

```
Enrollment(
    User: INT [PK] [FK to User.UserID],
    Course: VARCHAR(255) [PK] [FK to Courses.CRN]
)
```