# University Career Fair

Project Report

Business Data Management


Guided By:
Prof. Debopriya Ghosh

Submitted By:
Palak Patel
Adarsh Vijayaraghavan
Harshal Patel

# Contents

# I. Scope & System Description

We have attempted to design and build a database system for the complete tracking of a University Career Fair. We have used our experience from attending the career fairs conducted at Rutgers University to identify the use-cases, and have tried to completely model all requirements keeping in mind the standard software development practices.

The requirements and the scope of the system are as below:

1. Model the database for a career fair like the one hosted by Rutgers University Career Services
2. The career fair is conducted over a fixed time-frame, say 2 days. The duration is divided into 60-minute/30-minute slots.
3. Each organization has some representatives who add roles to the system
4. Some of these representatives visit the fair as recruiters
5. To do this, they must book time-slots at the fair on behalf of the organization, and be allocated to a stall
6. Organization representatives must also be able to individually rsvp for attending the career fair as recruiters.
7. An organization can post multiple roles at a time
8. Each role listing has some required skills-sets associated with it, along with the proficiency and prior experience expected of each skill. Each listing also has a list of expected majors.
9. Some roles may require specific language proficiencies which must be captured by the system
10. The system should persist the details of the candidates, including their skills, expertise, experience and the languages they speak
11. Candidates should be able to see roles matching their skill-set, and shortlist a few roles
12. Students must be able to rsvp for the career fair. The system must allow them to rsvp for partial attendance too.
13. Post the career fair, the organizations must be able to shortlist students
14. The system should be able to track basic information on the actual recruitment process, such as interview schedules
15. The system should allow various reporting capabilities, such as viewing the most popular skills, searching for roles and listing all organizations attending the career fair on a given day.
16. All user credentials captured in the system must follow standard security requirements
17. The system should be scalable, and must be capable of accommodating future changes without major restructuring of the Database
18. The system uniquely identifies administrators who are super users with additional privileges.

# II. Relationship Diagram

The ER diagram is shown below (Figure 1). This diagram was generated using MySQL Workbench.

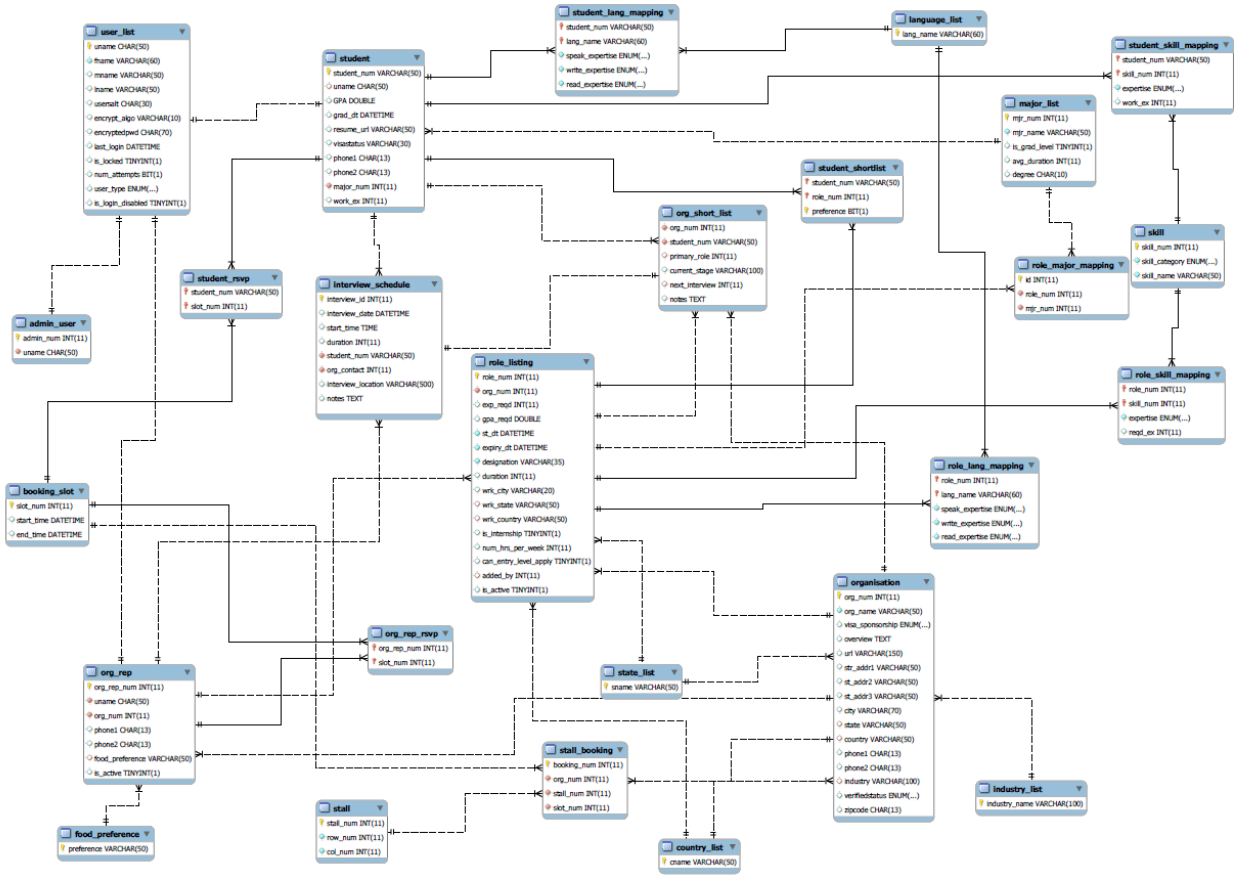


Figure 1: ER diagram

# III. Constraints supported by the data model

Constraints are used to define rules to allow or restrict the values that can be stored in columns. The purpose of introducing constraints is to enforce the integrity of a database. There are multiple ways to impose constraints, and our database model has used constraints extensively. We have explained a few samples of various constraints used in the system.

## 1. Participation constraint

A **participation constraint** defines the number of times an object in an object class can participate in a connected relationship set. Every connection of a relationship set must have a **participation constraint**.

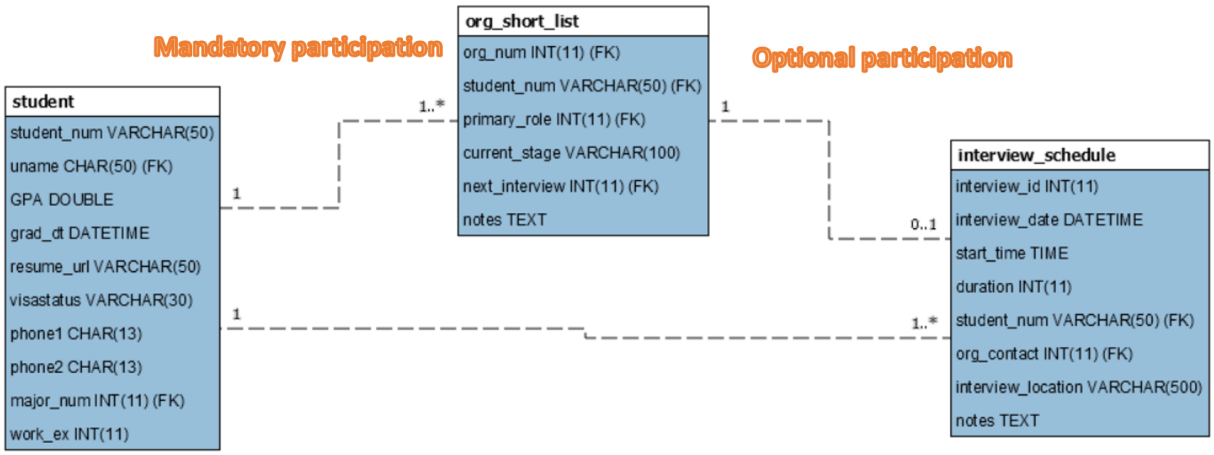


Figure 2: Example of participation constraint

An example of the usage of participation constraints is shown above (Figure 2). The *student* entity-set holds information of all students. The *org_short_list* entity-set stores the list of shortlisted students from each organization. The *interview_schedule* entity-set contains the details of interview schedules of shortlisted candidates.

When an organization shortlists a candidate, they obviously know the student they are shortlisting. Hence the *student_num* column in *org_short_list*, which is a foreign key referring to *student_num* from student, has Mandatory participation. However, an organization that short lists a student may not be immediately aware of the interview schedule. Also, consider the situation where an organization is impressed with a candidate but does not have a suitable role at the moment. In such cases, they can shortlist a student without an interview date. Hence the *next_interview* column in *org_short_list*, which refers to the *interview_id* column in *interview_schedule*, has optional participation.

## 2. Integrity Constraint

Integrity constraints provide a way of ensuring that changes made to the database by authorized users do not result in a loss of data consistency. Integrity constraints can be of many form; ranging from restricting the data-type and domain of a column to ensuring that there are no orphan foreign key records referring to non-existent primary keys from another table.

We have shown a couple of examples from our system where we make use of the ENUM data-type provided by MySQL to restrict the values that can be inserted in a column. We would like to restrict the values in the *skill_category* column of the *skill* entity set (Figure 3) to accept only the following values: 'Technical', 'Domain', 'Behavioural', 'Communication', 'Leadership', 'Other'. This is done by making the datatype of *skill_category* column as ENUM.

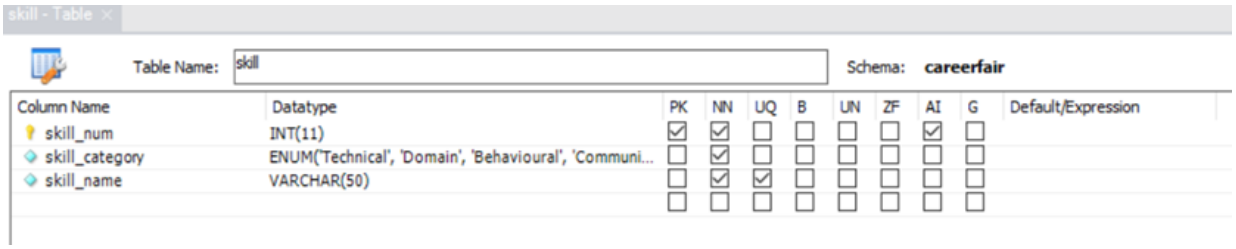


Figure 3: Use of ENUM in skill table

Another example is shown below (Figure 4), where we restrict the values allowed in *expertise* column of *student_skill_mapping* table to one of the following values: 'Expert', 'High', 'Moderate', 'Basic'.

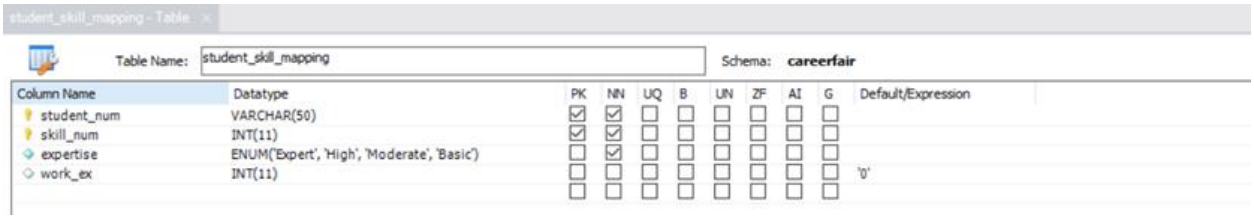


Figure 4: Use of ENUM in student_skill_mapping table

## 3. Cardinality Constraint

Cardinality is an important feature of entity-relationship models. In a relationship between two entity sets, cardinality specifies how many mappings a given entity from the first set can have with entities from the second set. Cardinalities are of the following kinds: one-to-one, one-to-many, and many-to-many.

Even though an ER diagram can represent a many-to-many relation, relations database management systems do not support many-to-many relationships as they would cause database redundancies. Instead, we break down many-to-many relations into two one-to-many relations using a bridge entity. Figure 5 shows an example of how this is done.

A *role_listing* record can map to multiple skills required for a role, and the same skill could be demanded by multiple *role_listing* records. Similarly, a *student* usually has more than one *skill*, and many students can have the same skill. We break these two many-to-many relationships using the bridge entities *role_skill_mapping* and *student_skill_mapping*. Given a single entity in the entity-set *student_skill_mapping*, it can refer to only one *skill* and one *student*. However, a given *skill* entity and a given *student* entity can occur multiple times in the *student_skill_mapping* entity-set. The *role_skill_mapping* entity-set works in a similar way.

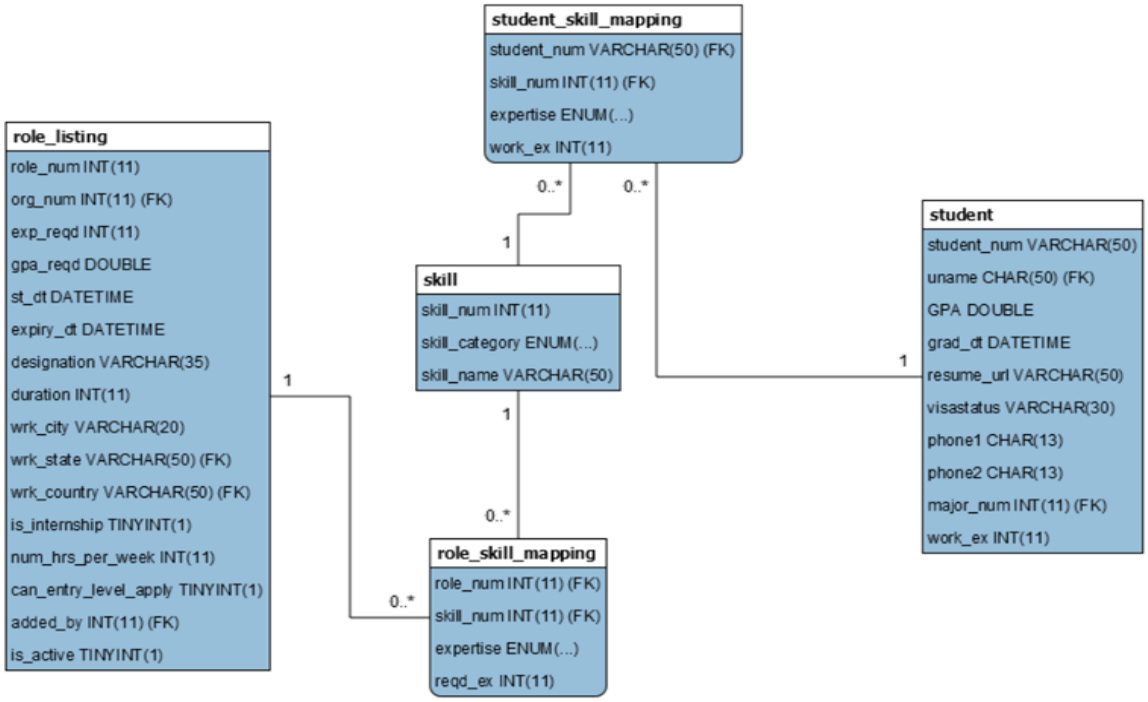


Figure 5: Example of cardinality constraints

# 4. Covering and Overlap Constraint

Generalization (or ISA hierarchy) is a technique used to reduce redundancy of data by identifying relations that have a parent-child relationship and extracting out the common attributes into a separate relation. For example, there are three kinds of users in the University Career Fair system: students, organization representatives, and administrators (who are part of the career services team). Since all users share certain common attributes such as name and user credentials, we have created a new entity-set *user_list* which holds these common attributes. As per standard design recommendation, the *user_list* entity-set has an attribute *user_type*, which is of type ENUM and identifies the specific subclass to which a given entity belongs to. This will help in joining two relations to get complete information about a given user. This is shown in Figure 6 below.

Covering constraint: within an ISA hierarchy, a covering constraint determines whether the entities in the subclasses collectively include all entities in the superclass. In our case, all users belong to either of the 3 categories we discussed. Hence, we can say that *student*, *admin* and *org_rep* COVER *user_list*.

Overlapping constraint: The overlap constraint determines whether the same entity can belong to more than one subclass entity-sets. In our case, a user can either be a student, an admin or an organization representative, but not more than one of the three groups. Hence, we disallow overlaps.

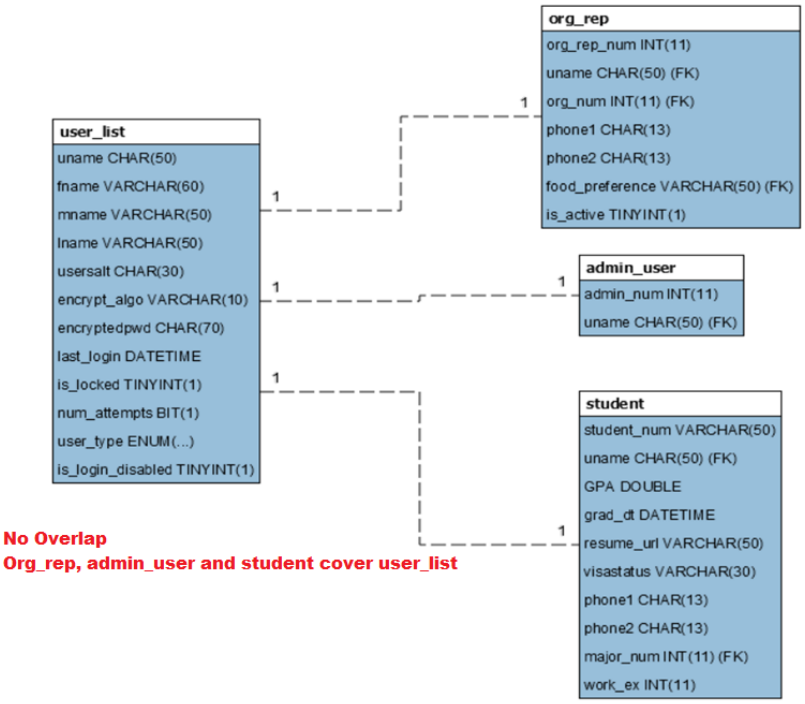


Figure 6: Covering and participation constraint

# IV. Data Dictionary

The data dictionary of each of the tables in our database system is shown below. Due to the sheer number of tables and columns, we have not described every column. We believe the naming of the columns are self-explanatory. Hence, we have just explained the columns that we feel are not very self-explanatory.

1. User_list

```
mysql> describe user_list;
+------------------+-----------------------------------+------+-----+---------+-------+
| Field            | Type                              | Null | Key | Default | Extra |
+------------------+-----------------------------------+------+-----+---------+-------+
| uname            | char(50)                          | NO   | PRI | NULL    |       |
| fname            | varchar(60)                       | NO   |     | NULL    |       |
| mname            | varchar(50)                       | YES  |     | NULL    |       |
| lname            | varchar(50)                       | YES  |     | NULL    |       |
| usersalt         | char(30)                          | YES  |     | NULL    |       |
| encrypt_algo     | varchar(10)                       | YES  |     | NULL    |       |
| encryptedpwd     | char(70)                          | YES  |     | NULL    |       |
| last_login       | datetime                          | YES  |     | NULL    |       |
| is_locked        | tinyint(1)                        | YES  |     | 0       |       |
| num_attempts     | bit(1)                            | YES  |     | b'0'    |       |
| user_type        | enum('STUDENT','ORG_REP','ADMIN') | YES  |     | NULL    |       |
| is_login_disabled | tinyint(1)                       | YES  |     | 0       |       |
+------------------+-----------------------------------+------+-----+---------+-------+
```

This is the generalized table that stores information about all users. As described in the section VII below, since we are storing user credentials in the tables, we have adhered to the OWASP security standards for password storage, which needs some additional columns for security.
The *encryptedpwd* field stores the encrypted user password. We do not store the plaintext password directly at any point of the time. When the user logs in to the system, the password they use to login is encrypted and then compared to stored password.
*usersalt* column stores the user specific salt. We need a salt for encryption. As per standard security practice, we break down the salt into two parts - one is a system wide salt used by the application and another is the user specific salt. This ensures that even if the application is compromised, a malicious agent will not have access to the user specific salt. And if the Database is compromised, the malicious agent is unlikely to have access to the application specific salt too. A unique salt is generated for each user using some deterministic algorithm while storing user details.
The *encrypt_algo* column stores the actual algorithm used for encryption. *Argon2* is currently the recommended algorithm for password encryption, but storing the algorithm in a user specific way gives the application a provision to change the algorithm on the fly. For instance, consider that the encryption methodology is being upgraded, but only for new users registering in the system. Such situations can be handled by this design.
The *last_login*, *is_locked*, *is_login_disabled* columns are used to handle Distributed Denial of Service (DDoS) attacks.
The user_type column is an enum data column that determines the subtype of the user.

2. student

```
mysql> describe student;
+-------------+-------------+------+-----+---------+-------+
| Field       | Type        | Null | Key | Default | Extra |
+-------------+-------------+------+-----+---------+-------+
| student_num | varchar(50) | NO   | PRI | NULL    |       |
| uname       | char(50)    | YES  | MUL | NULL    |       |
| GPA         | double      | YES  |     | NULL    |       |
| grad_dt     | datetime    | YES  |     | NULL    |       |
| resume_url  | varchar(50) | YES  |     | NULL    |       |
| visastatus  | varchar(30) | YES  |     | NULL    |       |
| phone1      | char(13)    | YES  |     | NULL    |       |
| phone2      | char(13)    | YES  |     | NULL    |       |
| major_num   | int(11)     | NO   | MUL | NULL    |       |
| work_ex     | int(11)     | YES  |     | NULL    |       |
+-------------+-------------+------+-----+---------+-------+
```

*uname* is a foreign key referring to *user_list* table. *major_num* is a foreign key referring to *major_list* table. *work_ex* is in months.

3. Org_rep

```
mysql> describe org_rep;
+-----------------+-------------+------+-----+---------+----------------+
| Field           | Type        | Null | Key | Default | Extra          |
+-----------------+-------------+------+-----+---------+----------------+
| org_rep_num     | int(11)     | NO   | PRI | NULL    | auto_increment |
| uname           | char(50)    | NO   | MUL | NULL    |                |
| org_num         | int(11)     | NO   | MUL | NULL    |                |
| phone1          | char(13)    | YES  |     | NULL    |                |
| phone2          | char(13)    | YES  |     | NULL    |                |
| food_preference | varchar(50) | YES  | MUL | NULL    |                |
| is_active       | tinyint(1)  | YES  |     | 1       |                |
+-----------------+-------------+------+-----+---------+----------------+
```

*uname* is a foreign key referring to *user_list* table. *org_num* refers to organisation table, and *food_preference* refers to *food_preference* table.

4. Admin_user

```
mysql> describe admin_user;
+------------+----------+------+-----+---------+----------------+
| Field      | Type     | Null | Key | Default | Extra          |
+------------+----------+------+-----+---------+----------------+
| admin_num  | int(11)  | NO   | PRI | NULL    | auto_increment |
| uname      | char(50) | NO   | MUL | NULL    |                |
+------------+----------+------+-----+---------+----------------+
```

*uname* refers to the *uname* column from *user_listing* table.

5. *Skill*

```
mysql> describe skill;
+----------------+-----------------------------------------------------------------------------------+------+-----+---------+----------------+
| Field          | Type                                                                              | Null | Key | Default | Extra          |
+----------------+-----------------------------------------------------------------------------------+------+-----+---------+----------------+
| skill_num      | int(11)                                                                           | NO   | PRI | NULL    | auto_increment |
| skill_category | enum('Technical','Domain','Behavioural','Communication','Leadership','Other')     | NO   |     | NULL    |                |
| skill_name     | varchar(50)                                                                       | NO   | UNI | NULL    |                |
+----------------+-----------------------------------------------------------------------------------+------+-----+---------+----------------+
```

6. Major_list

```
mysql> describe major_list;
+---------------+-------------+------+-----+---------+----------------+
| Field         | Type        | Null | Key | Default | Extra          |
+---------------+-------------+------+-----+---------+----------------+
| mjr_num       | int(11)     | NO   | PRI | NULL    | auto_increment |
| mjr_name      | varchar(50) | NO   |     | NULL    |                |
| is_grad_level | tinyint(1)  | YES  |     | NULL    |                |
| avg_duration  | int(11)     | YES  |     | NULL    |                |
| degree        | char(10)    | YES  |     | NULL    |                |
+---------------+-------------+------+-----+---------+----------------+
```

7. Language_list

```
mysql> describe language_list;
+-----------+-------------+------+-----+---------+-------+
| Field     | Type        | Null | Key | Default | Extra |
+-----------+-------------+------+-----+---------+-------+
| lang_name | varchar(60) | NO   | PRI | NULL    |       |
+-----------+-------------+------+-----+---------+-------+
```

8. Role_listing

*org_num* refers to the organisation table. *wrk_state* and wrk_country columns refer to the *state_list* and *country_list* tables. *added_by* refers to *org_rep* table.

```
mysql> describe role_listing;
+----------------------+-------------+------+-----+---------+----------------+
| Field                | Type        | Null | Key | Default | Extra          |
+----------------------+-------------+------+-----+---------+----------------+
| role_num             | int(11)     | NO   | PRI | NULL    | auto_increment |
| org_num              | int(11)     | NO   | MUL | NULL    |                |
| exp_reqd             | int(11)     | YES  |     | NULL    |                |
| gpa_reqd             | double      | YES  |     | NULL    |                |
| st_dt                | datetime    | NO   |     | NULL    |                |
| expiry_dt            | datetime    | NO   |     | NULL    |                |
| designation          | varchar(35) | NO   |     | NULL    |                |
| duration             | int(11)     | YES  |     | NULL    |                |
| wrk_city             | varchar(20) | YES  |     | NULL    |                |
| wrk_state            | varchar(50) | YES  | MUL | NULL    |                |
| wrk_country          | varchar(50) | YES  | MUL | NULL    |                |
| is_internship        | tinyint(1)  | YES  |     | NULL    |                |
| num_hrs_per_week     | int(11)     | YES  |     | NULL    |                |
| can_entry_level_apply | tinyint(1)  | YES  |     | NULL    |                |
| added_by             | int(11)     | YES  | MUL | NULL    |                |
| is_active            | tinyint(1)  | YES  |     | 1       |                |
+----------------------+-------------+------+-----+---------+----------------+
```

9. Organisation

```
mysql> describe organisation;
+-----------------+-----------------------------------------------------+------+-----+---------+----------------+
| Field           | Type                                                | Null | Key | Default | Extra          |
+-----------------+-----------------------------------------------------+------+-----+---------+----------------+
| org_num         | int(11)                                             | NO   | PRI | NULL    | auto_increment |
| org_name        | varchar(50)                                         | NO   |     | NULL    |                |
| visa_sponsorship | enum('Yes','No','Role-dependent')                  | YES  |     | NULL    |                |
| overview        | text                                                | YES  |     | NULL    |                |
| url             | varchar(150)                                        | YES  |     | NULL    |                |
| str_addr1       | varchar(50)                                         | YES  |     | NULL    |                |
| st_addr2        | varchar(50)                                         | YES  |     | NULL    |                |
| st_addr3        | varchar(50)                                         | YES  |     | NULL    |                |
| city            | varchar(70)                                         | YES  |     | NULL    |                |
| state           | varchar(50)                                         | YES  | MUL | NULL    |                |
| country         | varchar(50)                                         | YES  | MUL | NULL    |                |
| phone1          | char(13)                                            | YES  |     | NULL    |                |
| phone2          | char(13)                                            | YES  |     | NULL    |                |
| industry        | varchar(100)                                        | YES  | MUL | NULL    |                |
| verifiedstatus  | enum('NEW','TRUSTED','FLAGGED','BLOCKED','PENDING') | YES  |     | NULL    |                |
| zipcode         | char(13)                                            | YES  |     | NULL    |                |
+-----------------+-----------------------------------------------------+------+-----+---------+----------------+
```

State, country and industry refer to the master tables state_list, country_list and industry_list respectively. Note that the *verifiedstatus* column is of ENUM datatype and can be used to flag fraudulent organisations, and for devising a workflow to approve organisations that are newly added to the system.

10. State_list

```
mysql> describe state_list;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| sname | varchar(50) | NO   | PRI | NULL    |       |
+-------+-------------+------+-----+---------+-------+
```

11. Country_list

```
mysql> describe country_list;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| cname | varchar(50) | NO   | PRI | NULL    |       |
+-------+-------------+------+-----+---------+-------+
```

12. Industry_list

```
mysql> describe industry_list;
+---------------+--------------+------+-----+---------+-------+
| Field         | Type         | Null | Key | Default | Extra |
+---------------+--------------+------+-----+---------+-------+
| industry_name | varchar(100) | NO   | PRI | NULL    |       |
+---------------+--------------+------+-----+---------+-------+
```

13. *Role_skill_mapping*

```
mysql> describe role_skill_mapping;
+-----------+-----------------------------------------+------+-----+---------+-------+
| Field     | Type                                    | Null | Key | Default | Extra |
+-----------+-----------------------------------------+------+-----+---------+-------+
| role_num  | int(11)                                 | NO   | PRI | NULL    |       |
| skill_num | int(11)                                 | NO   | PRI | NULL    |       |
| expertise | enum('Expert','High','Moderate','Basic') | NO   |     | NULL    |       |
| reqd_ex   | int(11)                                 | YES  |     | NULL    |       |
+-----------+-----------------------------------------+------+-----+---------+-------+
```

*reqd_ex* is in months.

14. student_skill_mapping

```
mysql> describe student_skill_mapping;
+-------------+-----------------------------------------+------+-----+---------+-------+
| Field       | Type                                    | Null | Key | Default | Extra |
+-------------+-----------------------------------------+------+-----+---------+-------+
| student_num | varchar(50)                             | NO   | PRI | NULL    |       |
| skill_num   | int(11)                                 | NO   | PRI | NULL    |       |
| expertise   | enum('Expert','High','Moderate','Basic') | NO   |     | NULL    |       |
| work_ex     | int(11)                                 | YES  |     | 0       |       |
+-------------+-----------------------------------------+------+-----+---------+-------+
```

*student_num* refers to the *student_num* column from student table, and *skill_num* refers to the skill table. *work_ex* is in months.

15. Role_lang_mapping

```
mysql> describe role_lang_mapping;
+-----------------+-----------------------------------------+------+-----+---------+-------+
| Field           | Type                                    | Null | Key | Default | Extra |
+-----------------+-----------------------------------------+------+-----+---------+-------+
| role_num        | int(11)                                 | NO   | PRI | NULL    |       |
| lang_name       | varchar(60)                             | NO   | PRI | NULL    |       |
| speak_expertise | enum('Fluent','Proficient','Basic','None') | NO   |     | NULL    |       |
| write_expertise | enum('Fluent','Proficient','Basic','None') | NO   |     | NULL    |       |
| read_expertise  | enum('Fluent','Proficient','Basic','None') | NO   |     | NULL    |       |
+-----------------+-----------------------------------------+------+-----+---------+-------+
```

*role_num* column refers to the *role_listing* table, and *lang_num* refers to the *language_list* table.

16. Student_lang_mapping

```
mysql> describe student_lang_mapping;
+-----------------+-----------------------------------------+------+-----+---------+-------+
| Field           | Type                                    | Null | Key | Default | Extra |
+-----------------+-----------------------------------------+------+-----+---------+-------+
| student_num     | varchar(50)                             | NO   | PRI | NULL    |       |
| lang_name       | varchar(60)                             | NO   | PRI | NULL    |       |
| speak_expertise | enum('Fluent','Proficient','Basic','None') | NO   |     | NULL    |       |
| write_expertise | enum('Fluent','Proficient','Basic','None') | NO   |     | NULL    |       |
| read_expertise  | enum('Fluent','Proficient','Basic','None') | NO   |     | NULL    |       |
+-----------------+-----------------------------------------+------+-----+---------+-------+
```

*student_num* refers to the *student* table. *lang_name* refers to the *language_list* table.

17. role_major_mapping

```
mysql> describe role_major_mapping;
+----------+---------+------+-----+---------+----------------+
| Field    | Type    | Null | Key | Default | Extra          |
+----------+---------+------+-----+---------+----------------+
| id       | int(11) | NO   | PRI | NULL    | auto_increment |
| role_num | int(11) | NO   | MUL | NULL    |                |
| mjr_num  | int(11) | NO   | MUL | NULL    |                |
+----------+---------+------+-----+---------+----------------+
```

*role_num* refers to the *role_listing* table, and *mjr_num* refers to the *role_major_mapping* table.

18. Student_shortlist

```
mysql> describe student_shortlist;
+-------------+-------------+------+-----+---------+-------+
| Field       | Type        | Null | Key | Default | Extra |
+-------------+-------------+------+-----+---------+-------+
| student_num | varchar(50) | NO   | PRI | NULL    |       |
| role_num    | int(11)     | NO   | PRI | NULL    |       |
| preference  | bit(1)      | YES  |     | NULL    |       |
+-------------+-------------+------+-----+---------+-------+
```

The *preference* field is used to track and order the preference of each role for a student. Each student can shortlist up to a fixed number of roles. However, this must be handled from the application end as discussed in the "*Limitations*" (section VII) section below.
*student_num* refers to the *student* table and *role_num* refers to the *role_listing* table.

19. booking_slot

```
mysql> describe booking_slot;
+------------+----------+------+-----+---------+----------------+
| Field      | Type     | Null | Key | Default | Extra          |
+------------+----------+------+-----+---------+----------------+
| slot_num   | int(11)  | NO   | PRI | NULL    | auto_increment |
| start_time | datetime | YES  |     | NULL    |                |
| end_time   | datetime | YES  |     | NULL    |                |
+------------+----------+------+-----+---------+----------------+
```

When a new career fair is scheduled, the event dates will be broken down into timeslots, and one row for each timeslot will be inserted into this table. This table is used for booking slots, and for rsvp.

20) student_rsvp

```
mysql> describe student_rsvp;
+-------------+-------------+------+-----+---------+-------+
| Field       | Type        | Null | Key | Default | Extra |
+-------------+-------------+------+-----+---------+-------+
| student_num | varchar(50) | NO   | PRI | NULL    |       |
| slot_num    | int(11)     | NO   | PRI | NULL    |       |
+-------------+-------------+------+-----+---------+-------+
```

*student_num* refers to the *student_rsvp* table and *slot_num* refers to the *booking_slot* table.

21) org_rep_rsvp

This entity set holds the rsvp details of the organizational representatives who take part in the career fair as recruiters. While the *stall_booking* table (which we will discuss shortly) is for an *organisation* to book a *stall*, the representatives need to rsvp too. This will help the career services to know how many people are attending the fair and plan the logistics accordingly.

```
mysql> describe org_rep_rsvp;
+-------------+---------+------+-----+---------+-------+
| Field       | Type    | Null | Key | Default | Extra |
+-------------+---------+------+-----+---------+-------+
| org_rep_num | int(11) | NO   | PRI | NULL    |       |
| slot_num    | int(11) | NO   | PRI | NULL    |       |
+-------------+---------+------+-----+---------+-------+
```

22) stall

```
mysql> describe stall;
+-----------+---------+------+-----+---------+----------------+
| Field     | Type    | Null | Key | Default | Extra          |
+-----------+---------+------+-----+---------+----------------+
| stall_num | int(11) | NO   | PRI | NULL    | auto_increment |
| row_num   | int(11) | NO   | MUL | NULL    |                |
| col_num   | int(11) | NO   |     | NULL    |                |
+-----------+---------+------+-----+---------+----------------+
```

The *stall* table contains list of all *stalls* in the fair. The *row_num* and *column_num* columns refer to the location of the stall in the fair. It is assumed that the *row_num* and *stall_num* columns can together uniquely identify the physical location of the stall in a fair.

23) stall_booking

```
mysql> describe stall_booking;
+-------------+---------+------+-----+---------+----------------+
| Field       | Type    | Null | Key | Default | Extra          |
+-------------+---------+------+-----+---------+----------------+
| booking_num | int(11) | NO   | PRI | NULL    | auto_increment |
| org_num     | int(11) | NO   | MUL | NULL    |                |
| stall_num   | int(11) | NO   | MUL | NULL    |                |
| slot_num    | int(11) | NO   | MUL | NULL    |                |
+-------------+---------+------+-----+---------+----------------+
```

The *stall_booking* entity-set is a ternary relation between *booking_slot*, *organisation* and *stall* entity-sets. In case an organisation is visiting a career fair for 5 time-slots (each of 60/30-minute durations), there will be 5 records in this table.

24) org_shortlist

This entity set stores the information about the shortlisted candidate from each organisation. *org_num* refers to the *organisation* table. *student_num* refers to the *student* table. *primary_role* refers to the *role_listing* table. It is NULLable because an organisation should be able to shortlist candidates even if they don't have a matching role at the moment. The *next_interview* column refers to the *interview_schedule* table which we will discuss below. Once again, this is a NULLable column for reasons we described in section # III.

```
mysql> describe org_short_list;
+----------------+--------------+------+-----+---------+-------+
| Field          | Type         | Null | Key | Default | Extra |
+----------------+--------------+------+-----+---------+-------+
| org_num        | int(11)      | NO   | MUL | NULL    |       |
| student_num    | varchar(50)  | NO   | MUL | NULL    |       |
| primary_role   | int(11)      | YES  | MUL | NULL    |       |
| current_stage  | varchar(100) | YES  |     | NULL    |       |
| next_interview | int(11)      | YES  | MUL | NULL    |       |
| notes          | text         | YES  |     | NULL    |       |
+----------------+--------------+------+-----+---------+-------+
```

25) food_preference

```
mysql> describe food_preference;
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| preference | varchar(50) | NO   | PRI | NULL    |       |
+------------+-------------+------+-----+---------+-------+
```

This is a simple lookup table for storing food preferences. We could have used an ENUM datatype here. However we felt that new food preferences could be constantly added and removed, and that ENUM datatype would be inconvenient from a maintenance point of view.

26) inteview_schedule

```
mysql> describe interview_schedule;
+--------------------+--------------+------+-----+---------+----------------+
| Field              | Type         | Null | Key | Default | Extra          |
+--------------------+--------------+------+-----+---------+----------------+
| interview_id       | int(11)      | NO   | PRI | NULL    | auto_increment |
| interview_date     | datetime     | YES  |     | NULL    |                |
| start_time         | time         | YES  |     | NULL    |                |
| duration           | int(11)      | YES  |     | NULL    |                |
| student_num        | varchar(50)  | NO   | MUL | NULL    |                |
| org_contact        | int(11)      | NO   | MUL | NULL    |                |
| interview_location | varchar(500) | YES  |     | NULL    |                |
| notes              | text         | YES  |     | NULL    |                |
+--------------------+--------------+------+-----+---------+----------------+
```

The *interview_schedule* is used for tracking the actual interview process. *student_num* refers to the *student* table, and *org_contact* refers to the *org_rep* table. Both these columns are mandatory.

# V. Queries

This section describes some of the sample reports that can be generated using the University Career Fair system. All queries used here are provided as attachments, and they are also available in Section XI of this document.

## 1) What are the five most popular skills?

This is a very important query from analytics point of view. We use the GROUP BY clause to get the count of all role_listing for each class, and we use the LIMIT clause to limit the results to 5. The main table we refer to is *role_listing* table, with a join on *skill* table to get the name of the skill.

```
SELECT s.skill_name AS 'Skill',
COUNT(rs.role_num) AS '# Openings'
FROM role_skill_mapping rs
INNER JOIN skill s ON rs.skill_num = s.skill_num
GROUP BY (s.skill_name) LIMIT 5;
```

The results from executing the query is shown below.

| Skill | # Openings |
|---|---|
| Java | 2 |
| Procastination | 2 |
| Python | 2 |
| Smuggling | 2 |
| Spring | 2 |

## 2) What kind of food to order?

This query is useful from a logistics point of view for the career services team which organizes the career fair. We use a stored procedure *getFoodPreference* which takes in a date as parameter, and returns the number of recruiters attending the fair grouped by their food_preference.
We use the *org_rep_rsvp* table as the central table, with a join on *booking_slot* (for restricting the date).

```
# How much food of each kind should I order for recruiters who are coming today?
USE careerfair;

DROP PROCEDURE IF EXISTS getFoodPreferences;
DELIMITER $$
CREATE PROCEDURE getFoodPreferences
(IN fairdate datetime)
BEGIN
    SELECT o.food_preference AS 'Food Preference', COUNT(o.org_rep_num) AS '# Orders' FROM org_rep o
    INNER JOIN (SELECT DISTINCT(org_rep_num) FROM org_rep_rsvp ors WHERE ors.slot_num IN (
        SELECT b.slot_num FROM booking_slot b WHERE DATE(b.start_time) <= DATE(fairdate) AND
        DATE(b.end_time) <= DATE(fairdate))) AS rsvpd_rep
    ON rsvpd_rep.org_rep_num = o.org_rep_num
    GROUP BY o.food_preference
    ORDER BY o.food_preference;
END$$

SET @inDate = '2017-12-13 12:00:00';

CALL getFoodPreferences(@inDate);
```

Some test results from executing the query are shown below.

| Food Preference | # Orders |
|---|---|
| American | 1 |
| Mexican | 1 |
| Vegan | 1 |

3. Which organisations or visiting on the first day, but not the second?

The stored procedure *orgDiffBwTwoDays* takes in two different dates as input parameters, and returns the list of organizations that are visiting the career fair on the first day, but not the next. Such a query could be useful for the candidates to plan their schedule. We use the *stall_booking* table as the central table, with a join on *organisation* table. Ultimately, the query is a MINUS query, which is achieved in MySQL with the NOT IN operator.

```
# Which organizations that visit today are not visiting tomorrow?
USE careerfair;

DROP PROCEDURE IF EXISTS orgDiffBwTwoDays;
DELIMITER $$
CREATE PROCEDURE orgDiffBwTwoDays
(IN day1 datetime, IN day2 datetime)
BEGIN
    SELECT o.org_name AS 'Organisation', o.overview AS 'Overview', o.url AS 'Website',
    o.industry AS 'Industry', o.verifiedstatus
    AS 'Verified Status' FROM organisation o INNER JOIN
    (SELECT DISTINCT(org_num) AS 'org_num' FROM stall_booking WHERE slot_num IN
    (SELECT b.slot_num FROM booking_slot b WHERE DATE(b.start_time) = DATE(day1)) AND
    org_num NOT IN
    (SELECT DISTINCT(org_num) AS secondOrgNum FROM stall_booking WHERE slot_num IN
    (SELECT b.slot_num FROM booking_slot b WHERE DATE(b.start_time) = DATE(day2)))
    ) AS reqd_org_num
    ON o.org_num = reqd_org_num.org_num
END$$

SET @inDate1 = '2017-12-13 12:00:00';
SET @inDate2 = '2017-12-14 12:00:00';
CALL orgDiffBwTwoDays(@inDate1, @inDate2);
```

An example result from executing the stored procedure is shown below.

| Organisation | Overview | Website | Industry | Verified Status |
|---|---|---|---|---|
| Tesla | Tesla, Inc. (formerly Tesla Motors) is an American automaker, energy storage co... | https://www.tesla.com/ | Automotive | TRUSTED |

## 4. What's my interview schedule?

This is another useful query for candidates who can quickly lookup their interview schedules, Te stored procedure *getInterviewSchedule* takes in the student name (first, middle and last) as input. In a real world scenario, such a procedure can simply be called using the primary key, *student_num*. The calling application would in most cases be aware of the *student_num*, since the feature would probably be used by a student who is already logged in to the system. However we wanted to showcase the complexity of handling possible NULL values in SQL. We use some basic control statements to check if any of the parameters are empty space, and convert them to NULL. It is very likely that a calling application passes an empty string instead of NULL, and SQL treats these two differently.
We then use the <=> operator provided by SQL, which acts as both "=" and "IS" operator. This is because any of the parameters could either be NULL or have some value. The actual query is on the *interview_schedule*, and we perform some joins to get auxiliary information.

```
# Which organizations that visit today are not visiting tomorrow?
USE careerfair;

DROP PROCEDURE IF EXISTS getInterviewSchedule;
DELIMITER $$
CREATE PROCEDURE getInterviewSchedule
(IN fname varchar(60), IN mname varchar(50), IN lname varchar(50))
BEGIN
    #SELECT * FROM role_listing
    IF fname = '' THEN
        SET fname = null;
    END IF;
    IF mname = '' THEN
        SET mname = null;
    END IF;
    IF lname = '' THEN
        SET lname = null;
    END IF;

    SELECT concat(u.fname, ' ', u.lname) AS 'Student', i.interview_date AS 'Date', i.start_time AS 'Start Time',
    i.duration AS 'Duration (hrs)',     i.interview_location AS 'Location', i.notes AS 'Address',
    u2.fname + ' ' + u2.lname AS 'Organization contact', o.phone1 AS 'Contact ph. #', o.phone2  AS 'Contact Alt. Ph. #'
    FROM student s JOIN user_list u ON u.uname = s.uname AND NULLIF(u.fname,'') <=> fname
    AND NULLIF(u.mname,'')<=> mname AND NULLIF(u.lname,'') <=> lname
    JOIN interview_schedule i ON i.student_num = s.student_num
    JOIN org_rep o ON o.org_rep_num = i.org_contact
    JOIN user_list u2 ON u2.uname = o.uname;
END$$

SET @fname = 'Adarsh';
SET @mname = '';
SET @lname = 'Vijayaraghavan';
CALL getInterviewSchedule(@fname, @mname, @lname);
```

A sample result set of the stored procedure is shown below.

| Student | Date | Start Time | Duration (hrs) | Location | Address | Organization contact | Contact ph. # | Contact Alt. Ph. # |
|---|---|---|---|---|---|---|---|---|
| Adarsh Vijayaraghavan | 2018-01-01 00:00:00 | 13:00:00 | 2 | Central Park. New York | | 0 | 1232522656 | NULL |
| Adarsh Vijayaraghavan | 2018-02-01 00:00:00 | 13:00:00 | 2 | Plainsboro | | 0 | 1232522656 | NULL |

## 5. Show me some active roles.

This SQL query is used to fetch a list of all open roles. We use role_listing as the primary table, and join with *role_skill_mapping*, *role_language_mapping* and *role_major_mapping*. We also

use *skill* and *major_list* table for getting the names of the skills and majors. The challenge here is to group by roles, but still display all the languages, skills and majors required by each role. So, even though we display every role in a single row, the role might actually need multiple skills, which are available as multiple rows in the *role_skill_mapping* table. We solve this challenge by using the GROUP_CONCAT function.
We use LEFT JOIN because some roles may not be mapped to skills or languages. Such roles are assumed to be non-restrictive.

```
# Show active roles
USE careerfair;

SELECT r.designation AS 'ROLE', MAX(r.gpa_reqd) AS 'GPA' , MAX(r.st_dt) AS 'START DATE',
MAX(CONCAT(r.duration div 12, ' years and ',r.duration div 12, ' months'))  AS 'DURATION', MAX(r.wrk_city) AS 'City',
MAX(r.wrk_state) AS 'STATE', MAX(r.wrk_country) AS 'Country',
MAX(CASE r.is_internship WHEN 1 THEN 'YES' ELSE 'NO' END) AS 'Internship?', MAX(r.num_hrs_per_week) AS '# hrs/week',
MAX(CASE r.can_entry_level_apply  WHEN 1 THEN 'YES' ELSE 'NO' END) AS 'Open for entry level?',
GROUP_CONCAT(DISTINCT(s.skill_name) SEPARATOR '; ') AS 'Skills', GROUP_CONCAT(DISTINCT(rsm.expertise) SEPARATOR '; ') AS
'Expertises (in order)',
GROUP_CONCAT(DISTINCT(CONCAT(rsm.reqd_ex div 12, ' years and ',rsm.reqd_ex div 12, ' months')) SEPARATOR '; ') AS 'Exp Reqd (in
order)',
GROUP_CONCAT(DISTINCT(rlm.lang_name) SEPARATOR '; ') AS 'Languages',  GROUP_CONCAT(DISTINCT(m.mjr_name) SEPARATOR '; ') AS
'Majors'
FROM role_listing r
JOIN role_skill_mapping rsm ON r.role_num = rsm.role_num
    LEFT JOIN role_lang_mapping rlm ON rlm.role_num = r.role_num
    LEFT JOIN role_major_mapping rmm ON rmm.role_num = r.role_num
    LEFT JOIN skill s ON s.skill_num = rsm.skill_num
    LEFT JOIN major_list m ON m.mjr_num = rmm.mjr_num
    GROUP BY r.designation
    HAVING MIN(r.is_active) = 1;
```

A sample output from the stored procedure is shown below.

| ROLE | GPA | START DATE | DURATI | City | STATE | Country | Internsh | # hrs/we | Open for entry level? | Skills | Expertises (in order) | Exp Reqd (in order) | Languages | Majors |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ESPIONAGE AGENT | 3 | 2018-02-1 00:00:00 | 2 years and 2 months | Baltimore | Maryland | United States of America | NO | 50 | YES | Smuggling; Team player | Expert; Moderate | 0 years and 0 months | Chinese; English; Russian | NULL |
| OFFICE ASSISTANT | 3 | 2018-02-1 00:00:00 | 2 years and 2 months | Newark | New Jersey | United States of America | YES | 40 | YES | Positive Thinking; Procastinatior | Expert | 0 years and 0 months | NULL | NULL |
| SOFTWARE ENGINEER | 3.7 | 2018-01-1 00:00:00 | 2 years and 2 months | San Fransisco | California | United States of America | NO | 40 | YES | Java; Python; Spring | Expert; High; Basic | 0 years and 0 months; 1 years and 1 months | English | Computer Science; Information Technology |

# VI. Assumptions

We made the following assumptions while building the system.

1) An organization can book only one stall at a fair for a given time-slot. The same organisation can not occupy more than one slot.
2) Students can shortlist up to a fixed number of roles, say 10. This constraint will have to be implemented in the application layer. Implementing such a constraint in SQL will be tedious, and any mechanism that we use, such as trigger, is bound to have a impact on system performance. Also, a good software development practise is to separate out all

business logic to the service layer. Since this is a business logic, it makes sense to expect the service layer to handle the restriction.

3) The system need not capture information from an organization's perspective. For eg., historic information of interviews conducted by an organization can not be retrieved easily. The application is to assist the University in conducting the career fair. Though the organizations can add roles, view shortlisted candidates and so on, they can not track historical information in the system.
4) An organization may short-list students independent of role. A primary role may be optionally mentioned. Technically, *primary_role* in *org_short_list* table is a NULLable column
5) A company belongs only to one primary industry.
6) The row_num and column_num columns in the stall table hold two numbers that can uniquely identify the physical location of a stall at the fair. We can probably insert values such as latitude and longitude in these columns (in which case the column names and datatypes may have to be modified).

# VII. Limitations

Though we have put in considerable effort to make the system comprehensive, we recognize that our system has a few limitations, and has scope for enhancements. Some of the major ones we identified are listed below :

1) Care has been taken to use Enum datatype only for columns where a change of domain is unlikely. However if there is a change, modifying the Enum in a Production system would be relatively complex.
2) Some integrity constraints have not been handled by the Database, and need to be implemented in the application layer. One such example is with the *preference* column we discussed in section VI above.
3) Auto-increment values used as Primary Key columns in a few tables. In case an application connects to the database using a Object Relational Mapping (ORM) framework and Hi-Lo algorithm, this will cause a performance degradation.

# VIII. Unique Value Proposition

We feel our system design has certain unique features which we have highlighted below :

1) User credentials stored as per Open Web Application Security Project (OWASP) recommendations. The OWASP standard is an industry wide standard used globally for securing web applications.
2) The system can be scaled up to a career portal, where candidates and recruiters can find matches throughout the year. There are already some features in the system in this direction, such as flagging of fraudulent organizations, accommodation internships as well as full-time roles, storing interview schedules.
3) Use of Enum datatypes for selected columns to impose integrity constraints

# IX. Future Scope

This is not a finished product. We will be constantly working towards enhancing the system to accommodate more features. However we feel that this is a solid foundation to start with. Some of the enhancements we are planning to make to the system are listed below :

1) Create a service layer that abstracts the database through REST API calls.
2) Create a web based front end for the application that makes use of the REST service-layer.
3) Develop mobile friendly applications for popular mobile OSs
4) Enhance the database to meet the requirements of a career portal.

# X. References


1) Raghu Ramakrishnan and Johannes Gehrke. 2014. Database Management Systems (3nd ed.). Osborne/McGraw-Hill, Berkeley, CA, USA.
2) OWASP Password Storage Cheat Sheet : https://www.owasp.org/index.php/Password_Storage_Cheat_Sheet
3) MySQL Reference Manual : https://dev.mysql.com/doc/refman/5.7/en/


# XI. List of Attachments

1) Database Diagram : DBDiagramv2.4b.pdf

DBDiagramv2.4b.pdf

2) DDL.sql – script used for creating of all required tables

DDL.sql

3) DataInsert.SQL – script used for inserting test data into various tables.

DataInsert.sql

4) Set of all sample queries described in section V

SampleQueries.zip