Project #1

Xiaoran Ni

April 22, 2018

1 Introduction

In the first part of the project, the goal is to design a database that can serve as a relational backend for a new social networking site called Jobster that focuses on graduating students looking for jobs.

Specifically, the site should allow students to sign up and post resumes and other information. Companies can also sign up and post information about themselves and job announcements. Students can decide to follow certain companies (which means they will get all job announcements by those companies), can apply for jobs, and can friend other students and forward tips and job announcements to their friends. Companied can search for suitable applicants, and students can search for desirable jobs.

All the data about students, companies, jobs, connections, and notifications must be stored in a relational database, and a web frontend will be designed in the second part of this project to access the database and present the content to users in a nice form.

In this document, a detailed description of the design of the relational backend will be covered. To show the validation of such design, test data will be generated manually and results under such data on possible queries in practical usage will be provided.

2 Schema Design

2.1 Description

The overall design is illustrated in Figure 1, which is the ER diagram of it. Detailed information and logic of this design are as follows.

Basic entities The three basic entities are STUDENT, COMPANY, JOB. Considering this is a job searching website:

• For STUDENT, personal information (name) and educational information (university, major, GPA) are attributes, resume is also stored as a large file. Here an assumption that only one resume is stored in the website for each student is made. Because different students may have the same name, and index on

student can be anticipated to be used quite often in further queries, a unique id is generated for each student as he signs up on the website first time. Also, for login validation, a unique login name and a password must be resigned by a student and thus saved. In addition, keywords specified by students are stored.

• For COMPANY, a unique id is generated for each company as it signs up for the same reason. Each company have a unique company name and a password must be set up. Other attributes are the company name , the location of the company and the industry the company is in.

• For JOB, a unique id is generated for each job. The time when a job is posted in stored. Other attributes are the company id, the job location, the job title, the salary of the job, the background desired in this job and a detailed description of it.

Relational entities Here relational entities mean entities involve a student and a company, or two different students, or a job and a student of a company. Notification entities are a part of relational entities, though to make things more clear, they are treated as a independent category here.

• FOLLOW stores the information of the relationship between students and companies, a student can follow many companied or none, a company can be followed by many students or no one, hence FOLLOW is treated as a entity. Id of the student , id of the corresponding company and the time when the student follows this specific company are stored. A constraint here is a student can only follow a company once.

• To provide the function that students can be friend to each other, FRIEND is created. Two ids of the students involved and the time they become friends are stored. And this is a mutual relationship, in actual storage, considering a student with id_1 and a student with id_2, when they become friends at time t, both tuple (id_1, id_2, t) or (id_2, id_1, t) are stored in stead of storing one of them, though this method is not space efficient, querying about friends of a given student can be computed more efficiently.

• Students can message each other, MESSAGE entity stores the ids of the student who send the message and the one receive it, together with the content of the message. A student may message another student several times, such that the time of the message is also stored, and an assumption that only one message is allowed at a time is made.

• One of the most important relational entity for this website is the APPLY entity, which stores the id of the student applies, the job id of the job he applies to, the time he applies and the result of this application. The result value can be 'Under view', 'Interviewing', 'Failure' or 'Success'.

Notification entities The notification is designed in a way that after a user views a notification, this notification is no longer marked to him with a sign in the front end, but it is still stored in the database. Hence, a field status with a value of 'view' or 'unview' is designed as an attribute of each of them.

• Students can be friends to each other, for two students to become student, one of them have to send a request to the other, when the other student give a confirmation on this request, they become friends, such that a new tuple with their ids and the time will be inserted to the FRIEND table. This process is stored to the tables: NOTIFY_FRIEND_REQUEST. For the first part, given student with id id_1 sending a friend request to student with id id_2 at time t_1, a tuple $(id_1, id_2, t_1, null,' unview',' view')$ is stored in NOTIFY_FRIEND_REQUEST, the filed for answer is null because student id_2 hasn't replied to this request yet; the answer_status is 'view' hence this notification won't be marked on the home page of student id_1. Meanwhile, a notification pops on the main page of the student id_2. After the student id_2 clicked some button to see the detail of this notification, $(id_1, id_2, t_1, null,' unview',' view')$ is changed to $(id_1, id_2, t_1, null,' view',' view')$, and it will no longer show up on the main page of student id_2. Student id_2 can choose to apply to this request or just disregard. If he choose to be friend of student id_1, a button with 'yes' is pressed by him at time t_2, in the mean time, the tuple $(id_1, id_2, t_1, null,' view',' view')$ in NOTIFY_FRIEND_REQUEST is changed to $(id_1, id_2, t_1,' yes',' view',' unview')$, additionally, tuples (id_1, id_2, t_2) and (id_2, id_1, t_2) are inserted to the relational entity FRIEND. The showing up process of such notification to student id_1 is similar.

• NOTIFY_APPLY is the entity to notify a company that a student has just applied to a job post by them. i.e. A company with id *cid* posted a job with id *jid*, when a student with id *sid* decides to apply to this job at time t, a tuple $(sid, jid, t,' Underview')$ is inserted to table APPLY, and a tuple $(sid, jid, cid,' unview')$ is inserted to NOTIFY_APPLY. The usage of 'unview' and 'view' of this table is similar to as described in NOTIFY_FRIEND_REQUEST and NOTIFY_FRIEND_ANSWER.

• NOTIFY_JOB is the entity to notify a student that a job is announced by a company they are currently following. i.e. When a company with id *cid* announces a job with id *jid* at time t, this job is inserted into JOB table first, and if a student with id *sid* is following this company, a tuple $(jid, sid, t, unview)$ is inserted into NOTIFY_JOB, the usage of 'unview' is similar as above. Here time attribute is the time a job is notified to a student, such that a job can be notified to a same student several times.

2.2 Entity and key relationship

Under the design described in Section 2, the schema of the database of job applying website for now is:

STUDENT(sid, sname, login_name, password, university, major, GPA, resume, keywords)
COMPANY(cid, cname, location, industry)
JOB(jid, cid, time, location, title, salary, background, description)
FOLLOW(sid, cid, time)
FRIEND(sid1, sid2, time)

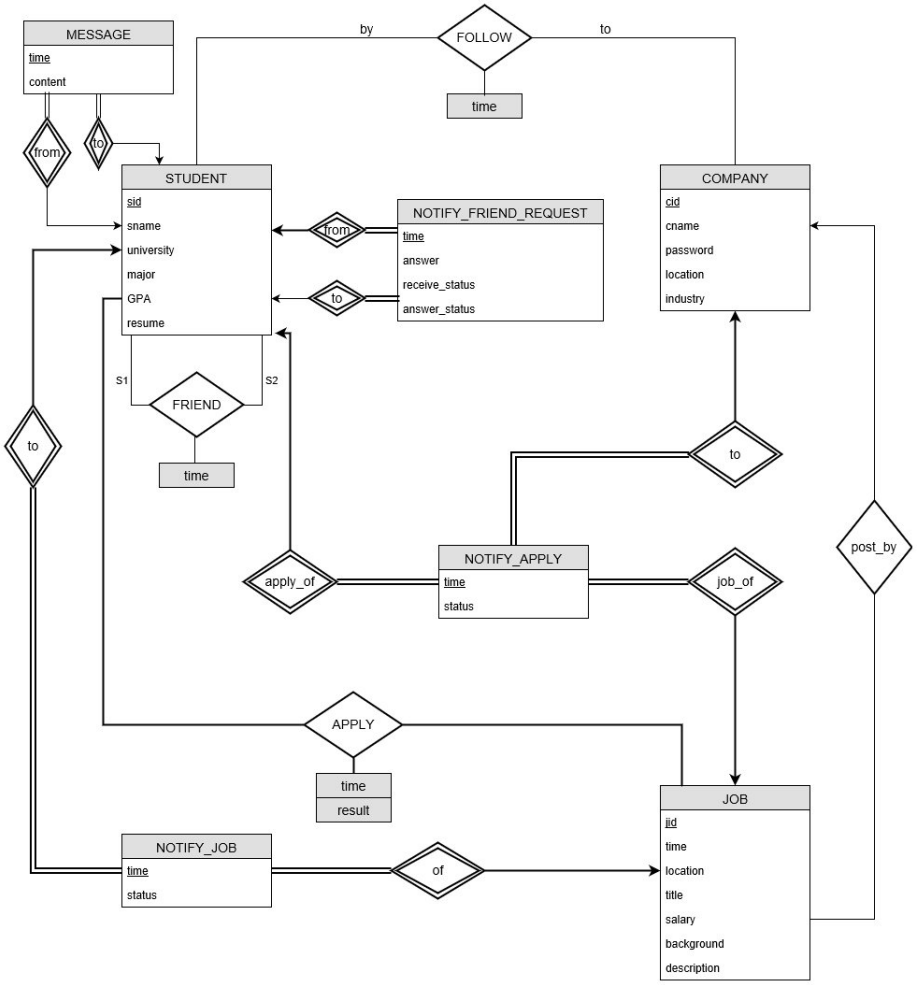

Figure 1: ER diagram of the project database

MESSAGE(from_sid, to_sid, time, content)
APPLY(sid, jid, time, result)
NOTIFY_FRIEND_REQUEST(from_sid, to_sid, time, answer, receive_status, answer_status)
NOTIFY_APPLY(sid, jid, cid, time, status)
NOTIFY_JOB(jid, sid, time, status)

The design consists of 11 tables currently. Primary keys are underlined in each table.
Foreign keys are:

JOB.cid is a foreign key referencing COMPANY.cid
FOLLOW.sid is a foreign key referencing STUDENT.sid
FOLLOW.cid is a foreign key referencing COMPANY.cid
FRIEND.sid1 is a foreign key referencing STUDENT.sid
FRIEND.sid2 is a foreign key referencing STUDENT.sid
MESSAGE.from_sid is a foreign key referencing STUDENT.sid
MESSAGE.to_sid is a foreign key referencing STUDENT.sid
APPLY.sid is a foreign key referencing STUDENT.sid
APPLY.jid is a foreign key referencing JOB.jid
NOTIFY_FRIEND_REQUEST.from_sid is a foreign key referencing STUDENT.sid
NOTIFY_FRIEND_REQUEST.to_sid is a foreign key referencing STUDENT.sid
NOTIFY_APPLY.(sid, jid) is a foreign key referencing APPLY.(sid, jid)
NOTIFY_JOB.jid is a foreign key referencing JOB.jid
NOTIFY_JOB.sid is a foreign key referencing STUDENT.sid

3 Test Data

The test environment is mysql 5.7.21, schema is created using attached file schema.sql, data is generated by file data.sql.

For testing the logic of the backend database, 6 sample queries are given in Section 4. Basic test data are generated for table STUDNET, COMPANY and JOB as each shown in Figure 2, 3, 4. 5 tuples of students are added to STUDENT, each consists of sid, sname, login_name, password, university, GPA and resume, sid is generated by database system incrementally and thus unique. The type for resume is set to varchar just to make testing simple, while in further parts of the project, a blob type is more suitable. 5 tuples of companies are added to COMPANY, each consists of cid, cname, password, location and industry, with cid is also generated automatically. 4 tuples are inserted into JOB, each consists of jid, cid, title, background and time, with jid created automatically. 3 tuples of JOB are created with date 2018-04-15, to test query 5.5 in Section 4, the rest one is set with the current date (2018-04-22).

sid	sname	login_name	password	university	GPA	resume
1	Bruce	Bruce01	abc135	NYU	4.0	database systems
2	Nancy	Nancy01	12345	NYU	3.7	database systems
3	Bruce	Bruce02	123	Cornell	3.7	database systems
4	Rogan	Rogan01	345	NYU	3.3	database systems
5	Amy	Amy01	abc	NYU	3.7	null

Figure 2: STUDENT table

To simulate the process of adding friends, three procedures are defined. Procedure add_friend is defined to add tuples to FRIEND given two student ids.

cid	cname	password	location	industry
1	Microsoft	mmm	New York	IT
2	Uber	uuu	New York	Transportation
3	Facebook	fff	Seattle	IT
4	TwoSigma	ttt	New York	Finance
5	Citibank	ccc	USA	Banking

Figure 3: COMPANY table

jid	cid	title	background	time
1	1	Software Engineer	MS in CS	2018-04-22
2	1	Machine Learning Engineer	MS in CS	2018-04-15
3	4	Data analyst	MS in Math	2018-04-15
4	3	Software Engineer	MS in CS	2018-04-15

Figure 4: JOB table

Produce request_friend is defined to add tuples to NOTIFY_FRIEND_REQUEST, Produce answer_friend is defined to modify both NOTIFY_FRIEND_REQUEST and to add tuples to FRIEND such that two students can become friends, which is described as in section 2. In test data, student with id 1 send request to student with id $2, 3, 4$, and all get an answer 'yes', such that 6 tuples are inserted into FRIEND and 3 tuples are inserted into NOTIFY_FRIEND_REQUEST as shown in Figure 5 and 6. Another 2 request are made from student 2 to student $3, 4$, with the first one neglected and the second one rejected, hence another 2 tuples are inserted into NOTIFY_FRIEND_REQUEST. In generation, all the request data are created with the date 2018-02-22, which is 2 months ago, to test query 5.3 in section 4. All the FRIEND tuples are created with the current time (2018-04-22).

sid1	sid2	time
1	2	2018-04-22
1	3	2018-04-22
1	4	2018-04-22
2	1	2018-04-22
3	1	2018-04-22
4	1	2018-04-22

Figure 5: FRIEND table

Procedure follow_company is created for a student to follow a company, which in practice can be called when a student pressed a 'Follow' button on their page. Using this procedure, 4 tuples are inserted into FOLLOW as in Figure 7.

from_sid	to_sid	time	answer	receive_status	answer_status
1	2	2018-02-22	yes	view	unview
1	3	2018-02-22	yes	view	unview
1	4	2018-02-22	yes	view	unview
2	3	2018-02-22	null	unview	view
2	4	2018-02-22	no	view	unview

Figure 6: NOTIFY_FRIEND_REQUEST table

sid1	cid	time
1	1	2018-04-22
1	2	2018-04-22
2	1	2018-04-22
3	1	2018-04-22

Figure 7: FOLLOW table

4 Query And Results

All the results are generated by using the test data described in Section 3. The results are shown in tables here, the snapshots of these query results can be found in attached file snapshots.

4.1 Create a record for a new student account, with a name, a login name, and a password

The SQL code is:

```
INSERT INTO STUDENT (sname, login_name, password)
        VALUES ('Bruce', 'Bruce01', 'abc135')
```

The result is shown in Figure 2, all the tuples are inserted using this code.

4.2 List the names of all friends of a particular user

Given a student with id user_id, the SQL code is:

```
SELECT S.sname
FROM FRIEND F JOIN STUDENT S
WHERE F.sid2 = S.sid AND F.sid1 = user_id
```

Given user_id = 1, the result is:

sname
Nancy
Bruce
Rogan

4.3 Delete all friendship requests that are older than one month and that have not been answered

The SQL code is:

```
DELETE FROM NOTIFY_FRIEND_REQUEST
WHERE answer is null
        AND timestampdiff(month, time, NOW()) >= 1
```

Using the NOTITY_FRIEND_REQUEST table with data in Figure 6, the result after deletion is:

from_sid	to_sid	time	answer	receive_status	answer_status
1	2	2018-02-22	yes	view	unview
1	3	2018-02-22	yes	view	unview
1	4	2018-02-22	yes	view	unview
2	4	2018-02-22	no	view	unview

4.4 List all students from NYU that are following Microsoft

The SQL code is:

```
SELECT S.*
FROM FOLLOW F, STUDENT S, COMPANY C
WHERE F.sid = S.sid AND F.cid = C.cid AND
        C.cname = 'Microsoft' AND S.university = 'NYU'
```

Using the following relationship data in Figure 7, the result is:

sid	sname	login_name	password	university	GPA	resume
1	Bruce	Bruce01	abc135	NYU	4.0	database systems
2	Nancy	Nancy01	12345	NYU	3.7	database systems

4.5 List all job announcements posted in the last week that are looking for some one with an MS in CS

The SQL code is:

```
SELECT *
FROM JOB
WHERE background LIKE '%MS in CS%' AND
        timestampdiff(week, time, NOW()) = 1
```

The result is:

jid	cid	title	background	time
2	1	Machine Learning Engineer	MS in CS	2018-04-15
4	3	Software Engineer	MS in CS	2018-04-15

4.6 For each student with GPA $>$ 3.5 whose resume contains the keyword "database systems", create a notification telling the student about a particular new job announcement that a company has posted

Given a job with id job_id, the SQL code is:

```
INSERT INTO NOTIFY_JOB (jid, sid, time, status)
        (SELECT job_id, sid, NOW(), 'unview'
        FROM STUDENT
        WHERE GPA > 3.5 AND resume like '%database systems%')
```

The result on the NOTIFY_JOB table is:

jid	sid	time	status
1	1	2018-04-22	unview
1	2	2018-04-22	unview
1	3	2018-04-22	unview

5 Evaluation

Each table defined as in subsection 2.2 is in BCNF, thus no redundant data is stored. The results from Section 4 can be validated easily, proving the correctness of the basic design of this backend system.

In this designation part, some attributes have been simplified. For instance, in STUDENT major should be a categorical variable, resume should be a blob type as a pdf format file might be stored in practice as mentioned above. There

might be other attributes not defined here, like email, phone number, which are quite important in job applying. In COMPANY, location can be divide into address, city, state and country, industry should be a categorical variable. In consideration of safety, password should be hashed in database system instead of storing it directly.

In practice, there can be other procedures that help manage the tables, e.g., when a notification has been viewed by a user, say a button is clicked, the status of corresponding tuple should be changed to 'view'; when a student applies for a job, a tuple should be inserted into NOTIFY_APPLY automatically; when a company post a job, tuples should be inserted into NOTIFY_JOB to notify the students who are currently following this company.

Access control is another problem, different accessibilities should be granted to students and companies. All these problems should be handled in the next part of this project.