Database and Information Systems Assignment

Benteng Ma 18206096

Yushi Wang 18206384

Fengbo Zhang 16206870

Mengxi Zhao 18206394

June 11,2021

Declaration of Authorship

We state that **we have full authorship for these whole pieces of work**, including this report, our video of demonstration, and the whole project we deliver, except from some of the methods and solutions we found through the internet, which, we will cite or refer in this report or in the program comments.

In addition, we state that **four of us share equal efforts and contributions**, though we've done different tasks, **everyone's work is irreplaceable**, and everyone shows strong willing of cooperation. Detailed tasks for each person will be given below in the report.

To show our confidence, and as a verification of our hard work, we would love to display everything we've learnt, thought and attempted during this course in this report.

We clearly understand the Academic Rules of Beijing University of Technology and University College Dublin, and we are aware of the punishment for violating the Rules of Beijing University of Technology and University College Dublin. We hereby promise to abide by the relevant rules and promise the originality of our work. If found violating the rules, we would accept the punishment thereof.

Menu

System Description

Our system is designed to be an online matchmaking (blind dating) system, with a very strong information system backend as support. During the implementation of this system, we covered almost every detail that was delivered during the lectures, including CURD language in MySQL that we can directly operate with in the system terminal, JDBC interface, newly presented Hibernate interface, information system designing methods, ER diagrams, mapping of it, and normalization of tables. The functions of this system will be shown below.

We may separate our system users as three kinds, **service users**, **psychological mentors** (customer staff), and also **system administrators** (system staff).

In our design, service users (will be named user here) are people who come to find their other halves. They provide their basic information as a key in the system, as well as their detailed attributes and characteristics, which will be used by the system to match the others. After registration, they are still allowed to modify their information. The users are recommended (**through a well-designed and exciting recommendation system**) by the system to each other (they are also able to search for others), they are able to see each other's provided information, and decide whether to "like" (focus on) each other. Both the users that is focusing on or be focused can know each other through the system. Users can also take part in events, they select them on a list, and they know where and when it happens. Additionally, we designed a type of mentors that support the users psychologically. Users are able to view them and select one for themselves. By design, they can even have relationship with another one, and have dates.

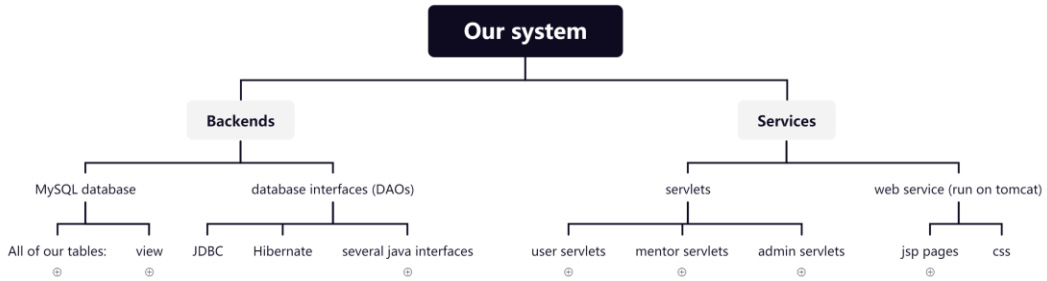
Psychological mentors are staffs of this system (employees), but from the system's view, they are also users, and they don't manage the system by themselves. As mentors, they are able to view the users that select them as mentor, and they are in charge of some locations, where they hold events, they can add one event location, and several events.

Administrators are the only type of people in this system that are able to delete people. They manage all the users, they can search and view all of them, they change the thing that users cannot change, and they can delete users.

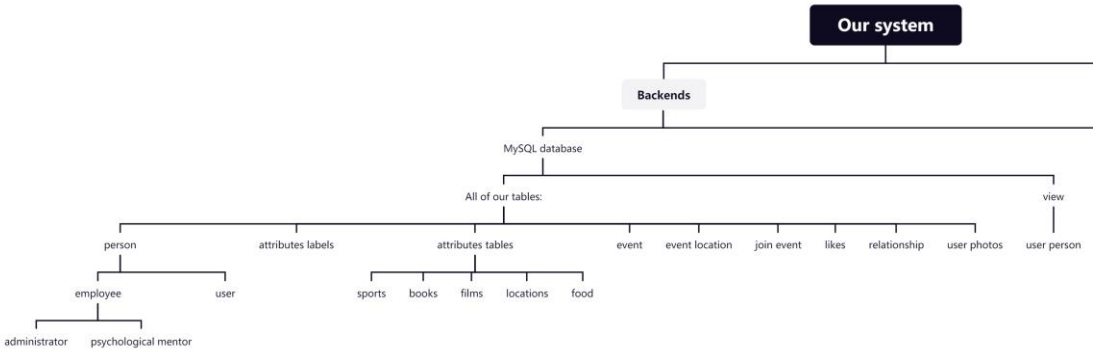
In the system, almost **all the relationships mentioned in class** are covered. We designed this sophisticated system to cover all of them, which will be shown in out ER diagram. Before that, we will introduce our system architecture, with diagrams.

System Architecture Diagram.

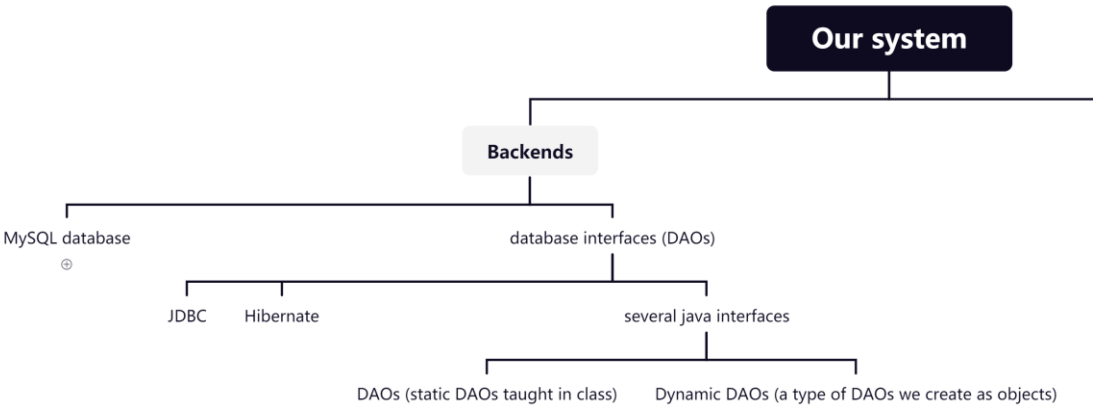
Our system consists of mainly the backend and service end. The back end system runs in parallel with the services.



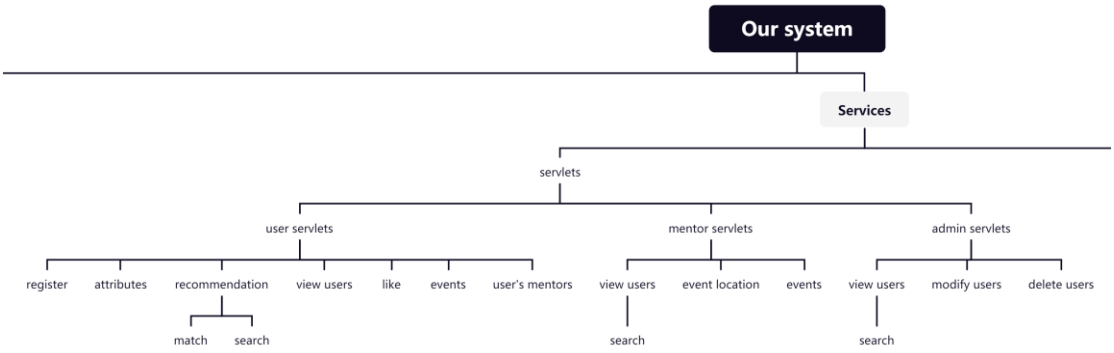
The database consists of several tables and a view.



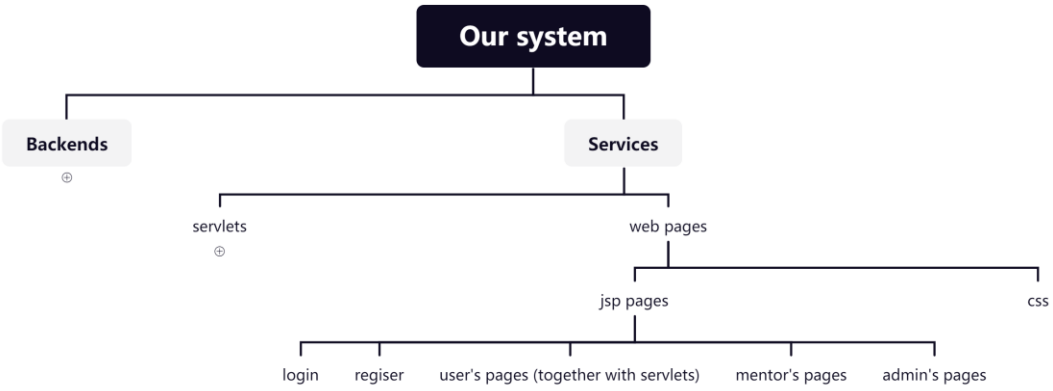
We used both JDBC and Hibernate for our database interface, and we wrote several DAOs.



We created several servlets to allow the communication between the webpages and backend, most of our functions are realized here.

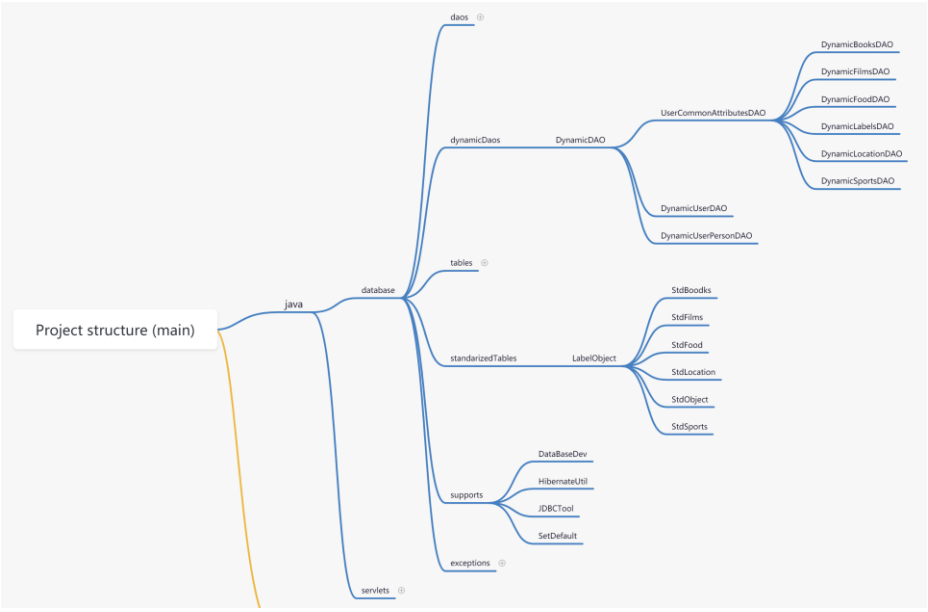


And finally, we have several webpages (supported with css) to provide user interface:

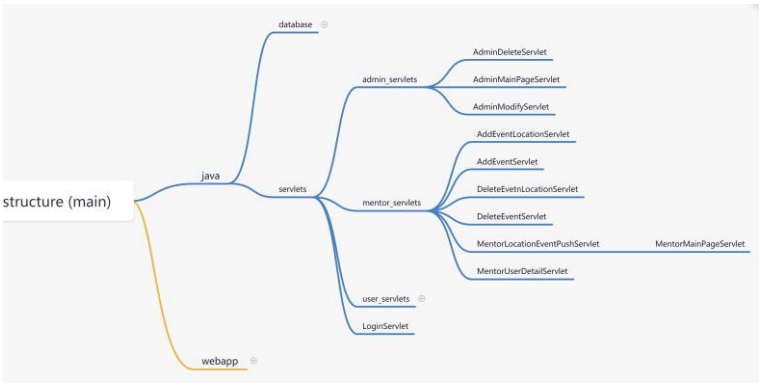


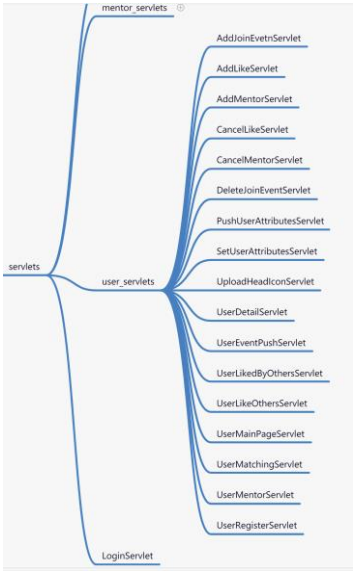
We will also introduce our project structure of code. Our code structure are strongly related with our system design, but they are still worthwhile to be introduced.

We would love to show specifically of the files with hierarchical structures.

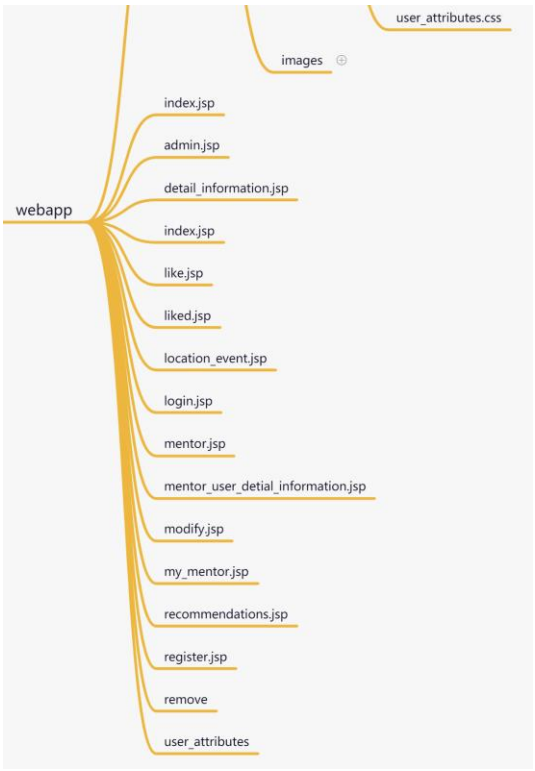


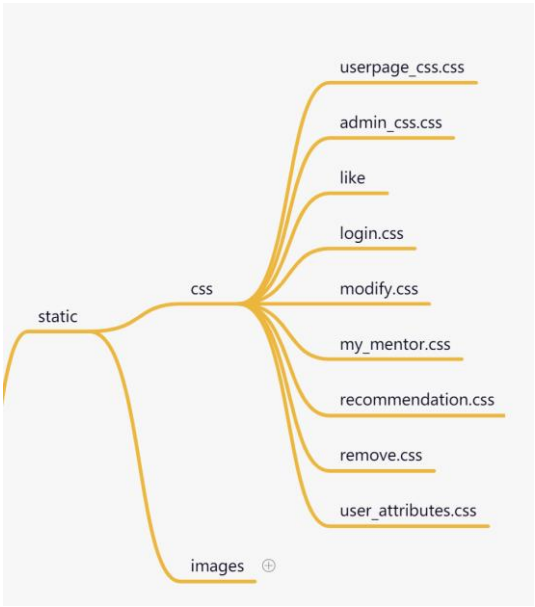
The servlets are shown below.





And also JSPs and CSSs.



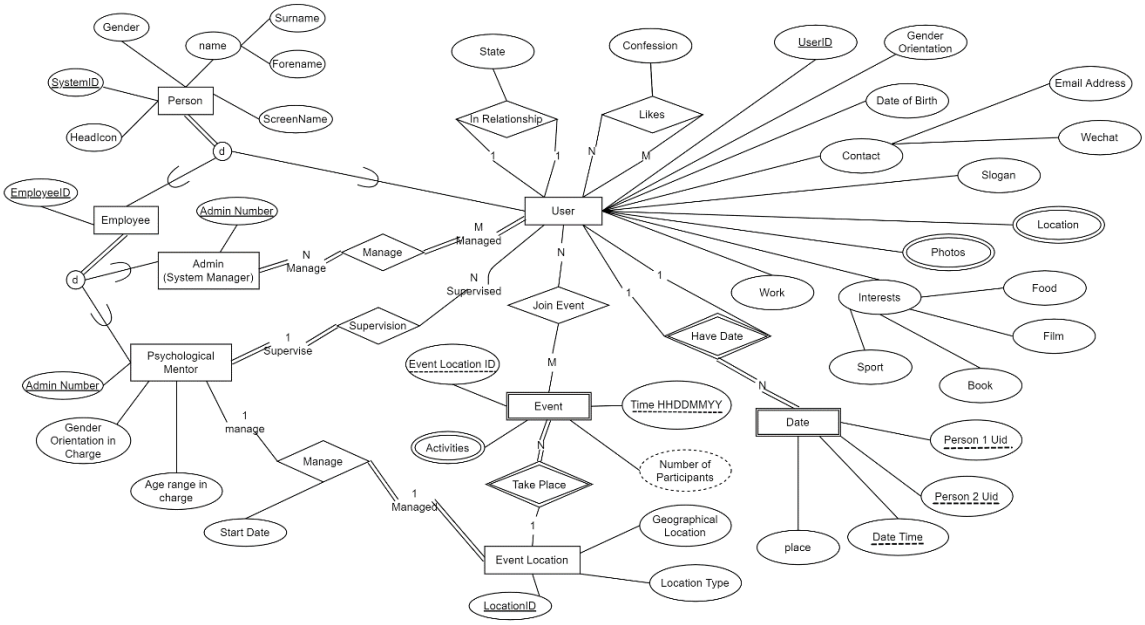


Tasks.

Before the real task, we made a demo project. The functions of the database and implementation in the demo are very simple. Moreover, we did not follow the complete process (like ER diagram, mapping…) to do the demo (we had not read all the lectures when we wrote the demo). **We put it in the appendix.** Although this is simple, it is the foundation of our project.

ER Diagram

The final ER diagram:

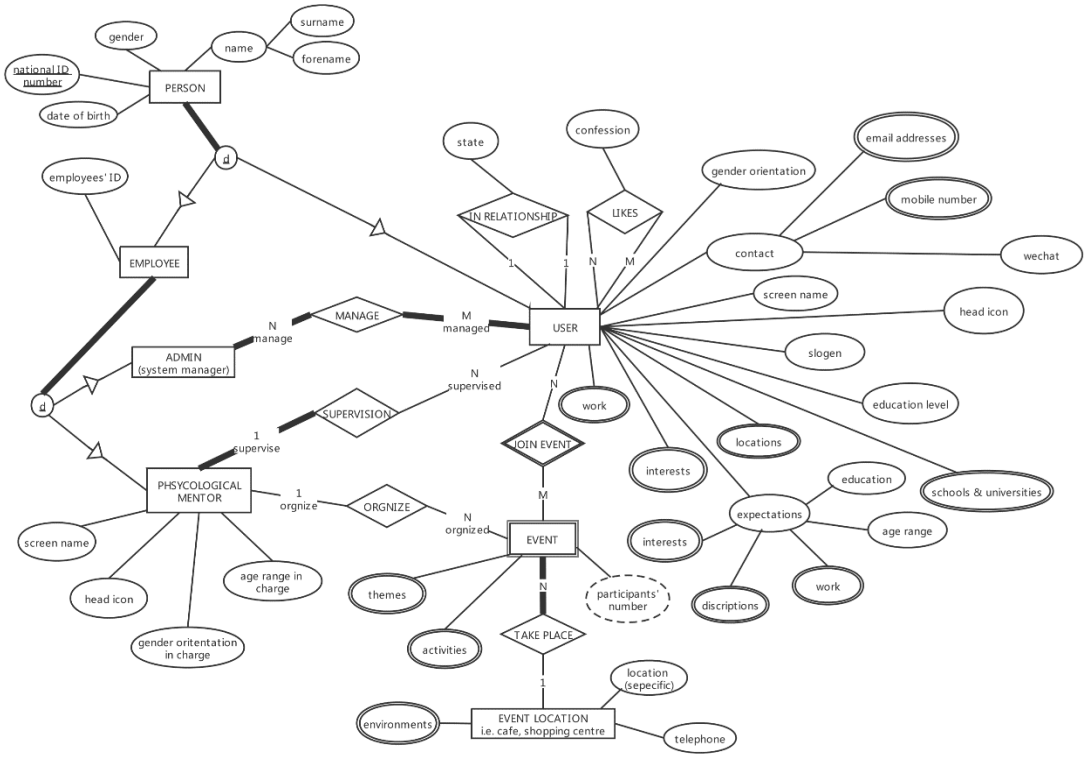


The process of ER diagram design.

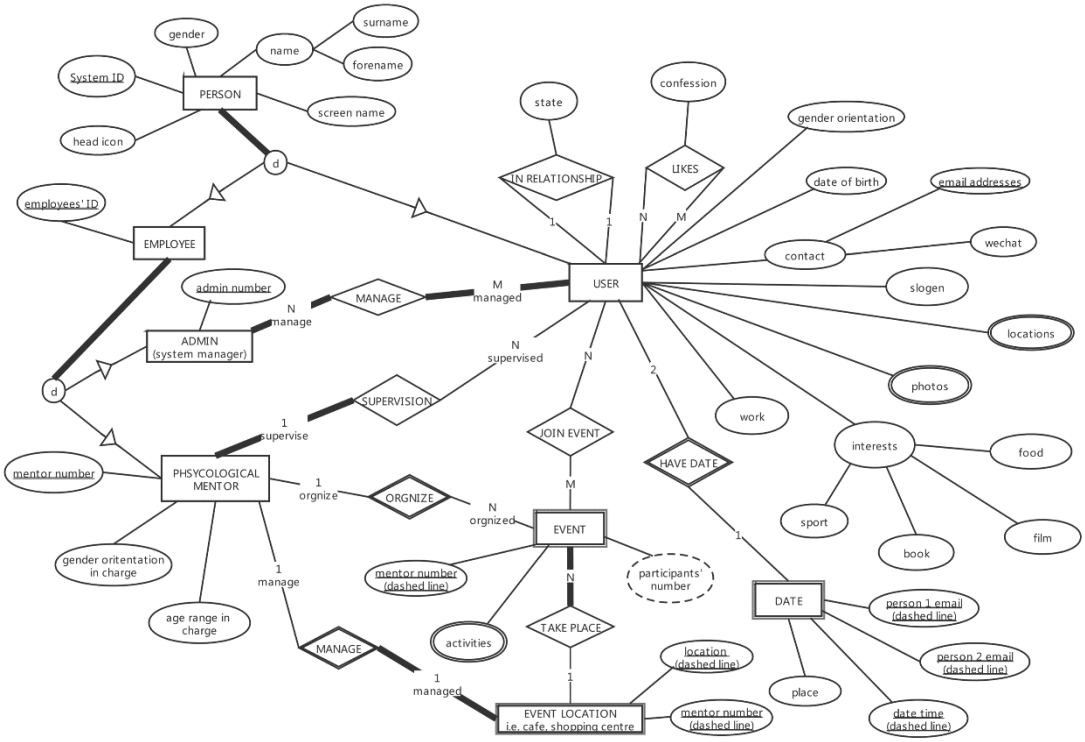
The original handwritten diagrams of the previous ER are shown in the appendix.

Around week12 (21 MAY) we created our first version ER diagram of our data system, and we improved it, eliminated it and get out second version. Our database will be mapped based on this second version ER diagram.

First version



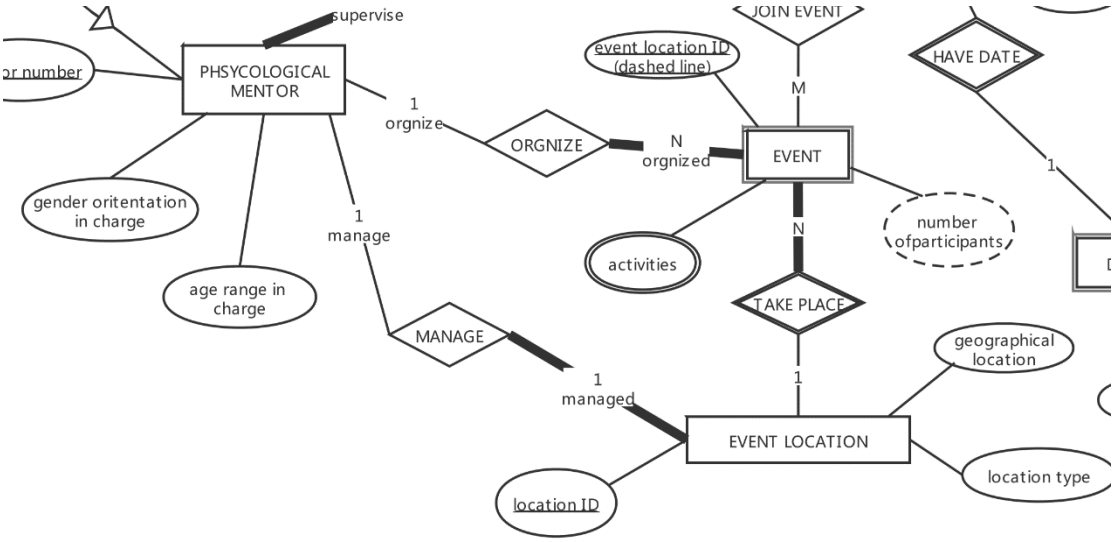
Second version



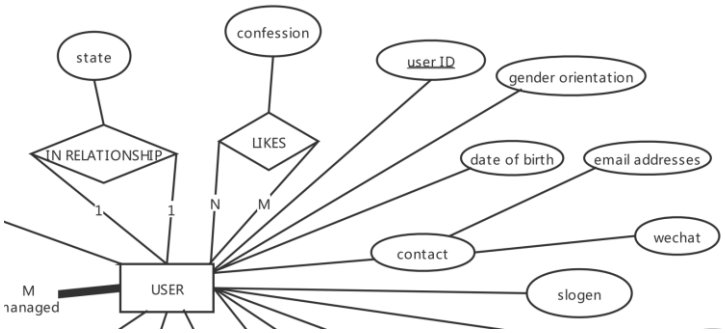
In the first version we designed hierarchical structure of three types of person as entities, which are system administrators, psychological mentors and users. Among which, admins and mentor are employees of this system, and the three of them all belong to person entity. We also prepared an event entity which is a weak entity depends on user.

In the second version we simplified some attributes from user entity; we also changed a weak entity - event to make it depend on mentor and decided to use mentor ID as its primary key; similarly, we also designed an event place entity which is also weak and depend on mentors; In addition, we add a date weak entity that uses users to be its primary key.

In the third version we simplified this event and event location system, redesigned some attributes and change the event attribute to depend on location, not mentor anymore. And since a location can be more static than a person – mentor, we changed it to be a normal entity not a weak one.



And in the fourth edition we add user ID back and use this as the primary key, not email address anymore.



We will map our database to this third version of ER diagram. And we will do this following the algorithm introduced in the lecture, except that we will consider sub-entities at the very beginning.

Tables

Mapping process

1. Mapping regular entity types (and consider the sub-entities)

PERSON

System ID	Surname	Forename	Gender	Screen name	Head icon

EMPLOYEE

System ID	Employee ID

ADMINISTRATOR

System ID	Employee ID	Admin number

PHSYCOLOGICAL MENTOR

System ID	Employee ID	Mentor number

Gender orientation in charge	Age range in charge

USER

System ID	User ID	Email address	Wechat	Gender orientation	Date of birth

Slogan	Locations (multiple values)	Photos (multiple values)	Work	Food	Film

Book	Sport

EVENT LOCATION

Location ID	Location type	Geographical location

2. Mapping weak entities

EVENT

Location ID	Time HHDDMMYY	Number of participants	Activities (multiple values)

DAT E

Uid1	Uid2	HHDDMMYY	Geographical location

3. Mapping 1: 1 relationship

MANAGE (some mentors manage location)
In EVENT LOCATION

Location ID	Location type	Geographical location	Manager ID	Start time DDMMYY

IN RELATIONSHIP and HAVE DATE
In USER

System ID	User ID	Email address	Wechat	Gender orientation	Date of birth

Slogan	Locations (multiple values)	Photos (multiple values)	Work	Food	Film

Book	Sport	Relation Uid	Relation state

4. Mapping 1: N relationships

SUPERVISION
In USER

System ID	User ID	Email address	Wechat	Gender orientation	Date of birth

Slogan	Locations (multiple values)	Photos (multiple values)	Work	Food	Film

Book	Sport	Relation Uid	Relation state	MENTOR ID

ORGNIZE
In EVENT

Location ID	Time HHDDMMYY	Number of participants (calculated through relationship)	Activities (multiple values)	MENTOR ID

5. Mapping M: N relationship

JOIN EVENT

Uid	EVENT LOCATION ID	TIME HHDDMMYY

LIKES

Uid1	Uid2	Confession

6. Mapping multivalued attributes

USER locations will be written as a string in designed format, so as EVENT activities.

USER PHOTOS

Uid	Photo path (in the system)

7. Mapping N-ary relationships

We've already cancelled all the relationships that have more than 2 types of entity referred, so we can skip this step.

Mapping supertype and subtypes

We've done this above; we keep the primary key of the supertype as a foreign key in the subtype.

Table design

Around week13 (27 MAY), with this ER diagram we would give our final design of database tables and we will give our normalized version of it later.

Week14, in the last version we used fixed strings (char) to record labels of users' likes such as food and sports. However, we now redesigned these into several M to N relationships, so that we can store infinite number of labels for each user, and it's easier for us to search persons with same labels.

PERSON

System ID	Surname	Forename	Gender	Screen name	Head icon	password
Auto increment integer	Varchar(64)	Varchar(64)	Enum('female', 'male', 'unselected')	Char(64) Unique	Varchar(128) (system paths for images)	Char(64) Not null

EMPLOYEE

System ID	Employee ID
Foreign key from PERSON	Auto increment integer

ADMINISTRATOR

System ID	Employee ID	Admin number
Foreign key from PERSON	Foreign key from EMPLOYEE	Auto increment integer

PHSYCOLOGICAL MENTOR

System ID	Employee ID	Mentor number
Foreign	Foreign	Auto inc int

Gender orientation in charge	Age range in charge begin	Age Range end
Enum('homosexual', 'hetero')	int	int

USER

System ID	User ID	Email address	Wechat	Gender orientation	Date of birth
Foreign	Auto inc int	Varchar(20)	Varchar(20)	Enum('homosexual', 'hetero') default 'hetero'	DATE

Slogan	Work	MENTOR ID
Varchar(20)	One number from Labels	Foreign key from MENTOR

*these are multiple value attributes in design, but in practical they store labels with maximum length of 3 (e.g., '02;'), so they are just chars in the DB.

*Book	*Sport	Relation Uid	Relation state
Char(30)	Char(30)	Foreign	Enum('acquaintance', 'ambiguous', 'boyfriend/girlfriend', 'engaged', 'married')
Moved to label object		Move to Relationship table	

Relation Uid and Relation state are removed later.

EVENT LOCATION

Location ID	Location type	Geographical location	Manager ID	Start time DDMMYY
Auto inc int	Enum('coffee /tea house', 'bar', 'restaurant', 'shopping centre', 'part', 'cinema', 'theatre')	Varchar(256)	Foreign	DATE

EVENT

Location ID	Time HHDDMMYY	Number of participants (calculated through relationship)	Activities (multiple values)
Foreign	DATETIME	Integer	Char(15)

The mentor here may not be the same person that manages this location

DAT E

Uid1	Uid2	HHDDMMYY	Geographical location
Foreign	Foreign	DATETIME	Varchar(256)

JOIN EVENT

Uid	EVENT LOCATION ID
Foreign	Foreign

LIKES

Uid1	Uid2	Confession
Foreign	Foreign	Varchar(512)

USER PHOTOS

Uid	Photo path (in the system)
Foreign	Varchar(128)

LABELS

Serial	Locations	Work	Food	Film	Book	Sport	Activity
Unique auto inc	Varchar(64)	--	--	--	--	--	--

Location

Lid	Uid

Food

Fid	Uid

Books

Bid	Uid

Films

Fid	Uid

Sports

Sid	Uid

Normalization process,

Turn these data design into normal forms.

The 1st normal form requires no repeating attributes. At present level of table design, we actually have one repeating attributes, which is the "state" of relationships. Although these should be 1-to-1 relationships, this attribute of "state" is actually kept in both users, but they are exactly same, so they are stored twice, and they need double modifications. According to the course slides, this can be a case of having a repeating group. To normalize this, we can add an extra table for this relationship, just like what we've done in "LIKES".

So, it should be:
RELATIONSHIP

Uid1	Uid2	Relation state
Foreign USER key	Foreign USER key	Enum('acquaintance', 'ambiguous', 'boyfriend/girlfriend', 'engaged', 'married')

Where we keep the second uid with greater value than the first; and we can remove these attributes from USER.

For 2NF, it is defined that, a relation is in 2NF if, and only if, it is in 1NF and every non-key attribute is fully functionally dependent on the whole key. For our case, we have several tables with more than one primary keys, and for them, we need to check each of the attributes, to see if they depend on all the primary keys, if not, we would take them out of the original table.

Our tables with more than one primary key are EVENT, DATE, JOIN EVENT, and LIKES. However, we checked them one by one and we think all the attributes in them must depend on all the primary keys. We've already considered these issues while we were designing our ER diagram and tables.

For 3NF, we searched for transitive functional dependencies. The only thing we found that might refer to this issue is that in EVENT LOCATION, it feels like the start time of a mentor's management should depend on this mentor. But consider a case that a mentor originally manages a place, and now he/she goes on a vacation, someone else take this job for a period, but then he/she returns, and starts to manage again. This example shows that the time is actually independent of mentor, so we don't need change anything for it. As a result, we are already in 3NF.

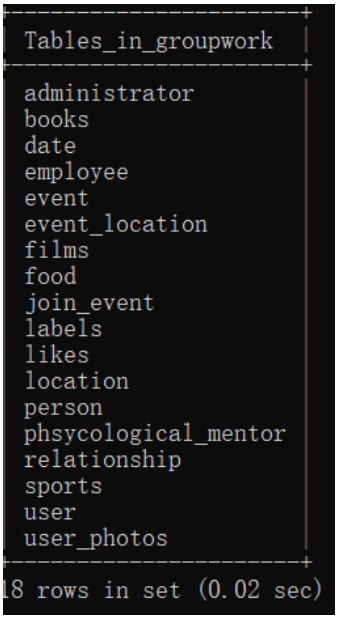
For BCNF, we need to solve issue of overlapping candidate keys. In our system, using USER table as an example, there are actually two determinant keys, user ID and system ID. But we already solve the issue of this overlapping dependency, by keeping user ID as the primary key, and letting all the local attributes depend on the user ID, not system ID. In addition, as it is shown in the ER diagram and the tables above, we've made a classification of attributes, for PERSON, EMPLOYEE, ADMIN, MENTOR and USER. They all have this determinant key issue originally, but we separate the shared attributes to the higher level entities, and cutoff such shared dependency, only keep the candidate keys as foreign keys (not primary keys). As a result, we may say that we are already in BCNF.

Create Table in MySQL, the specific statement is not shown here. See appendix for details.

CREATE VIEW UserPerson AS SELECT person.SystemID, person.Surname, person.Forename, person.Gender, person.ScreenName, person.HeadIcon, person.password, user.UserID, user.Emailaddress, user.Wechat, user.GenderOrientation, user.DataOfBirth, user.Slogan, user.Work, user.MentorID FROM user inner join person on user.SystemID = person.SystemID;

It's worth noticing that we wrote the VIEW and JOIN syntax at the end. We originally wanted to inherit the JOIN statement in JDBC, but later we realized it was more efficient to write the implementation in SQL. I'll make this clearer later in Hibernate and JDBC inheritance. So, what we need to explain here is that the VIEW and JOIN statement joins the two tables and view them as UserPerson. But since there are SystemID in both the User and Person tables, this attribute display duplicated in the View during table merging. Therefore, we must specify each attribute in the view. A view is a visual table based on the result set of an SQL statement. The view contains rows and columns, just like a real table. We can add SQL functions, WHERE, and JOIN statements to the view, like we did in UserPerson. We would use this view in DynamicUserPerson file later.

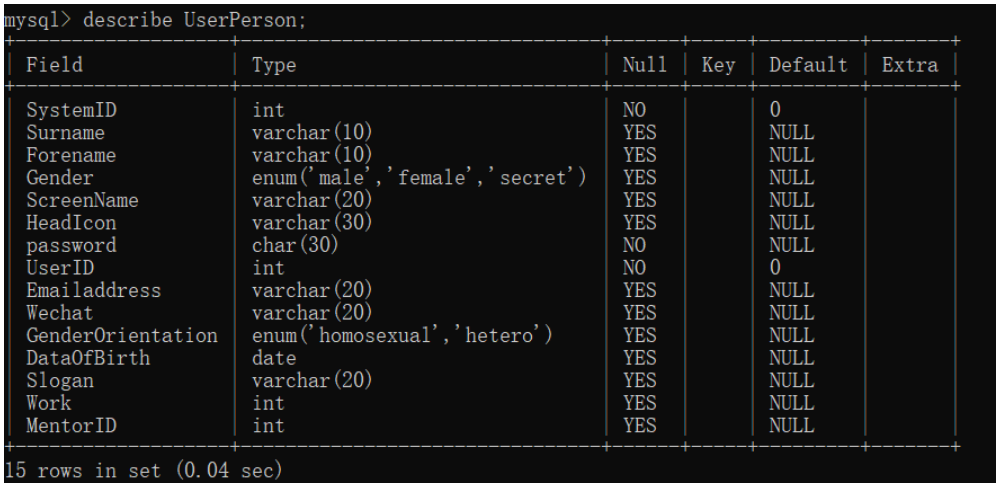
According to the above, established the database.



```
+-----------------------+
| Tables_in_groupwork   |
+-----------------------+
| administrator         |
| books                 |
| date                  |
| employee              |
| event                 |
| event_location        |
| films                 |
| food                  |
| join_event            |
| labels                |
| likes                 |
| location              |
| person                |
| phsycological_mentor  |
| relationship          |
| sports                |
| user                  |
| user_photos           |
+-----------------------+
18 rows in set (0.02 sec)
```

The specific table description is not shown here. See appendix for details.

The VIEW UserPerson is discrebed as below



```
mysql> describe UserPerson;
+-------------------+---------------------------------+------+-----+---------+-------+
| Field             | Type                            | Null | Key | Default | Extra |
+-------------------+---------------------------------+------+-----+---------+-------+
| SystemID          | int                             | NO   |     | 0       |       |
| Surname           | varchar(10)                     | YES  |     | NULL    |       |
| Forename          | varchar(10)                     | YES  |     | NULL    |       |
| Gender            | enum('male','female','secret')  | YES  |     | NULL    |       |
| ScreenName        | varchar(20)                     | YES  |     | NULL    |       |
| HeadIcon          | varchar(30)                     | YES  |     | NULL    |       |
| password          | char(30)                        | NO   |     | NULL    |       |
| UserID            | int                             | NO   |     | 0       |       |
| Emailaddress      | varchar(20)                     | YES  |     | NULL    |       |
| Wechat            | varchar(20)                     | YES  |     | NULL    |       |
| GenderOrientation | enum('homosexual','hetero')     | YES  |     | NULL    |       |
| DataOfBirth       | date                            | YES  |     | NULL    |       |
| Slogan            | varchar(20)                     | YES  |     | NULL    |       |
| Work              | int                             | YES  |     | NULL    |       |
| MentorID          | int                             | YES  |     | NULL    |       |
+-------------------+---------------------------------+------+-----+---------+-------+
15 rows in set (0.04 sec)
```

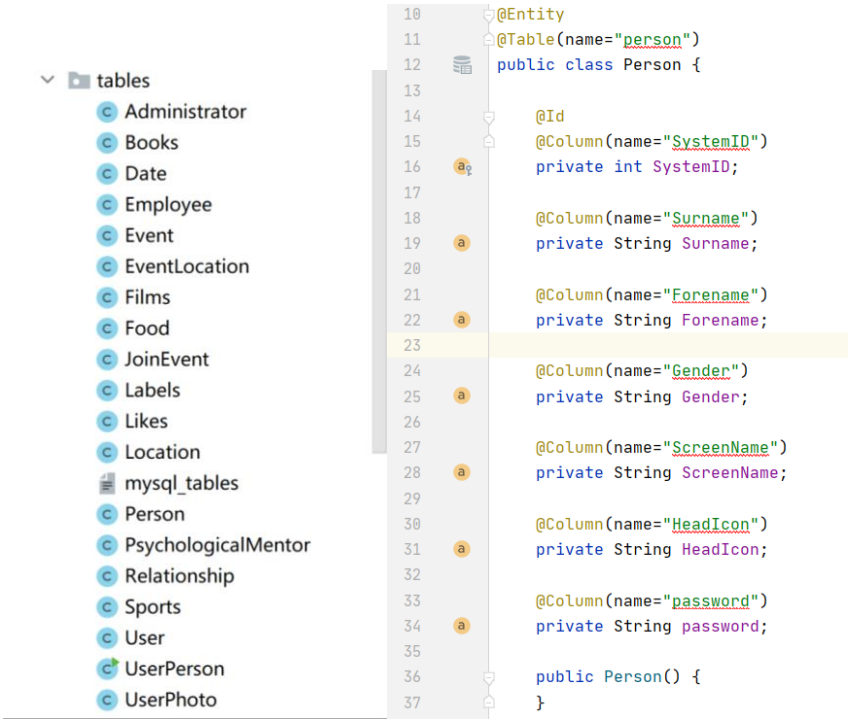
Table object and Data Access Objects (DAOs)

All the files in daos used Hibernate ORM, which explained in Week13. Compared with use sql method with Connection and Statement, using Hibernate did not need to call session every time. It saves a lot of time, especially when it needs to check a lot of data through one table. Hibernate is therefore reusable. Beyond that, using Hibernate brings convenience for us to write unity. For inserts and updates, Hibernate provides more freedom for writing, which Hibernate can directly return a created object.

HibernateUtil is the connect configuration between data access layer and JDBC. After we created one table object, we should add the configuration class into it. Due to the different time zones and fonts, there was a problem when configuring the Configuration that the font time zone could not be recognized. Later we used the Asian time zone, and although the way of display was correct, we found that we still had time zone problems. For example, in Beijing we want to add a person's birthday on April 27, and the date stored in JDBC is April 26. After changing to Shanghai time zone, the time becomes correct. (week 14, 6.2)

```
public class HibernateUtil {


    private static SessionFactory sessionFactory;

    public static SessionFactory getSessionFactory() {
        if (sessionFactory == null) {
            try {
                Configuration configuration = new Configuration();
                // Hibernate settings equivalent to hibernate.cfg.xml's properties
                Properties settings = new Properties();
                settings.put(Environment.DRIVER, "com.mysql.cj.jdbc.Driver");
                settings.put(Environment.URL, "jdbc:mysql://localhost:3306/groupwork?useSSL=false");
                settings.put(Environment.URL, "jdbc:mysql://localhost:3306/groupwork?serverTimezone=Asia/Shanghai&characterEncoding=utf-8");
                settings.put(Environment.USER, "root");
                settings.put(Environment.PASS, "premacybenny");
                settings.put(Environment.DIALECT, "org.hibernate.dialect.MySQL5Dialect");
                settings.put(Environment.SHOW_SQL, "true");
                settings.put(Environment.CURRENT_SESSION_CONTEXT_CLASS, "thread");
                configuration.setProperties(settings);
                configuration.addAnnotatedClass(Person.class);
                configuration.addAnnotatedClass(User.class);
                configuration.addAnnotatedClass(Employee.class);
                configuration.addAnnotatedClass(Administrator.class);
```

Mapping every table in Hibernate form. The example person is shown below.



```
tables
  Administrator
  Books
  Date
  Employee
  Event
  EventLocation
  Films
  Food
  JoinEvent
  Labels
  Likes
  Location
  mysql_tables
  Person
  PsychologicalMentor
  Relationship
  Sports
  User
  UserPerson
  UserPhoto
```

```
@Entity
@Table(name="person")
public class Person {

    @Id
    @Column(name="SystemID")
    private int SystemID;

    @Column(name="Surname")
    private String Surname;

    @Column(name="Forename")
    private String Forename;

    @Column(name="Gender")
    private String Gender;

    @Column(name="ScreenName")
    private String ScreenName;

    @Column(name="HeadIcon")
    private String HeadIcon;

    @Column(name="password")
    private String password;

    public Person() {
    }
```

Hibernate helps us persistently map into database tables, reducing the hassle of extracting information directly from the tables. From every table's DAO, it has 5 basic methods: get all rows from every table/ save one object to the table/ get one object by key/ delete object by key/ update one object.

The statement in Hibernate Query is HQL and the name in the createQuery function is the table

object, not the real table name as SQL. The slide show the function of getAllObjects in the table and create one object in the table, as below. After having done the function, close the session.

```
public static List<Sports> getAllSports() {
    List<Sports> l  = null;
    try (Session session = HibernateUtil.getSessionFactory().openSession()) {
        l = session.createQuery( s: "from Sports").list();
        session.close();
    } catch (Exception e) {
       e.printStackTrace();
    }
    return l;
}

public static void saveSports(Sports e) {
    Transaction transaction = null;
    try (Session session = HibernateUtil.getSessionFactory().openSession()) {
        transaction = session.beginTransaction();
        session.save(e);
        transaction.commit();
    } catch (Exception exp) {
        if (transaction != null) {
            transaction.rollback();
        }
        exp.printStackTrace();
    }
}
```

The basic function get one object by key is similar as get the list of objects. But the creatQuery function only return one unique result.

```
public static Sports getSportsByKey(int sid,int uid) {
    Sports l = null;
    try (Session session = HibernateUtil.getSessionFactory().openSession()) {
        l = (Sports)session.createQuery( s: "from Sports where Sid = "+ sid + " and Uid = "+uid).uniqueResult();
        session.close();
    } catch (Exception e) {
       e.printStackTrace();
    }
    return l;
}
```

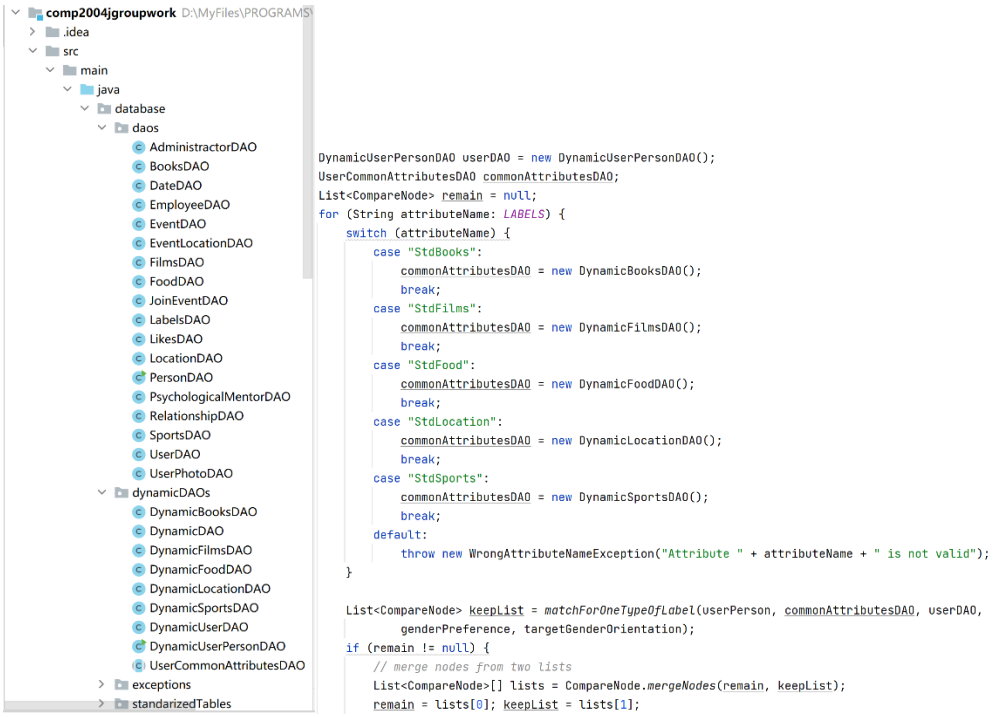
The delete and update operation is a little bit different. Create an object and add the primary key into the object and use Transaction to do operation of the table.

```
public static void deleteSportsByKey(int sid,int uid) {
    Sports l = new Sports();
    try (Session session = HibernateUtil.getSessionFactory().openSession()) {
        Transaction transaction=session.beginTransaction();
        l.setUid(uid);
        l.setSid(sid);
        session.delete(l);
        transaction.commit();
        session.close();
    } catch (Exception e) {
       e.printStackTrace();
    }
}
```

```
public static void updateSports(Sports e) {
    try (Session session = HibernateUtil.getSessionFactory().openSession()) {
        Transaction transaction=session.beginTransaction();
        session.update(e);
        transaction.commit();
        session.close();
    } catch (Exception m) {
       m.printStackTrace();
    }
}
```

The files in dynamicDAOs are re-designed version of DAOs above, which need one instance to

operate database. It is not static, and the sessions are only closed after finishing all the operations needed, which saves a lot of time for opening and closing sessions. It is worth mentioning that the method used in DynamicUserPerson is from Week6 lecture, sql statement in java. The reason is that in this function, we wanted to join the user table and person together but, hibernate do not support sql JOIN method. In this function, we used prepared statement as mentioned in the lecture. (week14 6.4)




```
public UserPerson getUserPersonBySystemID(int systemID) {
    try {
        String columns = "person.SystemID, person.Surname, person.Forename, person.Gender, person.ScreenName," +
                "person.HeadIcon, person.password, user.UserID, user.Emailaddress, user.Wechat, user.GenderOrientation, " +
                "user.DataOfBirth, user.Slogan, user.Work, user.MentorID";
        String sql = "SELECT " + columns + " FROM user inner join person on user.SystemID = person.SystemID";
        sql = "SELECT * FROM (" + sql + ") as UserPerson WHERE SystemID = ?;";
        PreparedStatement ps = connection.prepareStatement(sql);
        ps.setInt( parameterIndex: 1, systemID);
        System.out.println(ps);
        ResultSet rs = ps.executeQuery();
        rs.next();

        int userID = rs.getInt( columnLabel: "UserID");
        String emailAddress = rs.getString( columnLabel: "Emailaddress");
        String wechat = rs.getString( columnLabel: "Wechat");
        String genderOrientation = rs.getString( columnLabel: "GenderOrientation");
        Date dataOfBirth = rs.getDate( columnLabel: "DataOfBirth");
        String slogan = rs.getString( columnLabel: "Slogan");
        int work = rs.getInt( columnLabel: "Work");
        int mentorID = rs.getInt( columnLabel: "MentorID");
        String surname = rs.getString( columnLabel: "Surname");
        String forename = rs.getString( columnLabel: "Forename");
        String gender = rs.getString( columnLabel: "Gender");
        String screenName = rs.getString( columnLabel: "ScreenName");
        String headIcon = rs.getString( columnLabel: "HeadIcon");
        String password = rs.getString( columnLabel: "password");

        ps.close();
        UserPerson userPerson = new UserPerson(systemID, userID, emailAddress, wechat, genderOrientation, dataOfBirth,
                slogan, work, mentorID, surname, forename, gender, screenName, headIcon, password);
        return userPerson;
    } catch (SQLException e) {
```

However, we changed our opinion. Instead of creating a temporary sql String called UserPerson each time, we decided to build a VIEW called UserPerson directly in SQL using the View statement. And I showed how to do that in the previous section. Now I just need to select the expectation part from the view UserPerson.(week15 6.9)

```java
public UserPerson getUserPersonByMentorID(int mentorID) {
    try {
        String sql = "SELECT * FROM UserPerson WHERE MentorID = ?;";
        PreparedStatement ps = connection.prepareStatement(sql);
        ps.setInt( parameterIndex: 1, mentorID);
        System.out.println(ps);
        ResultSet rs = ps.executeQuery();
        rs.next();

        int systemID = rs.getInt( columnLabel: "SystemID");
        int userID = rs.getInt( columnLabel: "UserID");
        String emailAddress = rs.getString( columnLabel: "Emailaddress");
        String wechat = rs.getString( columnLabel: "Wechat");
        String genderOrientation = rs.getString( columnLabel: "GenderOrientation");
        Date dataOfBirth = rs.getDate( columnLabel: "DataOfBirth");
        String slogan = rs.getString( columnLabel: "Slogan");
        int work = rs.getInt( columnLabel: "Work");
        String surname = rs.getString( columnLabel: "Surname");
        String forename = rs.getString( columnLabel: "Forename");
        String gender = rs.getString( columnLabel: "Gender");
        String screenName = rs.getString( columnLabel: "ScreenName");
        String headIcon = rs.getString( columnLabel: "HeadIcon");
        String password = rs.getString( columnLabel: "password");

        ps.close();
        UserPerson userPerson = new UserPerson(systemID, userID, emailAddress, wechat, genderOrientation, dataOfBirth,
                slogan, work, mentorID, surname, forename, gender, screenName, headIcon, password);
        return userPerson;
    } catch (SQLException e) {
        e.printStackTrace();
        return null;
    }
```

IntelliJ IDEA 2020.2.4 available

Apart from JOIN statement which does not support by Hibernate, we also found some other disadvantages of Hibernate. Such as multiple mapping and deletion.

We created one interface LabelObject for all interests, like Sports, Books, Films, Food and traveled location. These table have one common characteristic that they only include their label ID and user ID and they takes two together as their primary key. With this inerface, many method of different classes can be written into one.

```java
package database.standarizedTables;

public interface LabelObject {
    public int getLabelId();
    public void setLabelId(int labelId);
    public int getUserId();
    public void setUserId(int UserId);
}
```

To control the objects successfully, we created Stdxxxx for every interested which implement LabelObject interface and use Hibernate interface with the original table. In every Std class, it could get every LabelID anf UserID, which is uniform and easy to manage.

In the subsequent process, we found that this created some inevitable problems. Such as when we want to get mutiple object though one judgement condition. e.g. find all users' ID which interested sport's ID = 5. It will return a lot of IDs but is repetitive. We checked a lot of time, and the grammar and logic is correct. After checking on the Internet, we know that if use multiple interfaces of Hibernate mapping, it will only displayed the first data queried repeatedly. After several fruitless modifications, we decided to go back to sql method with Connection and

Statement. And then we got a hopeful answer.(week 14 6.5)

```java
@Override
public List<LabelObject> getAllValuesWithUserID(int userID) {
    List<LabelObject> list = new LinkedList<>();
    try {
        PreparedStatement ps = connection.prepareStatement( sql: "SELECT * FROM sports WHERE Uid = ?;");
        ps.setInt( parameterIndex: 1, userID);
        ResultSet rs = ps.executeQuery();
        while (rs.next()){
            int Sid = rs.getInt( columnLabel: "Sid");
            int Uid = rs.getInt( columnLabel: "Uid");
            StdSports s = new StdSports(Sid,Uid);
            list.add(s);
        }
        rs.close();
        ps.close();
    } catch (Exception e) {
        e.printStackTrace();
        list = null;
    }
    // for (LabelObject each: list) System.out.println((each.getLabelId()));
    return list;
}
```

Hibernate deleting objects is also a serious problem. If the object being deleted has a NULL value in the table, an error will be recognized during deletion and the deletion will fail. The solution is the same as above. Use both JDBC Tool and Hibernate to solve the error part. (Week 15 6.9)

Similar with Hibernate, change the time zone in JDBC tool into Shanghai zone.

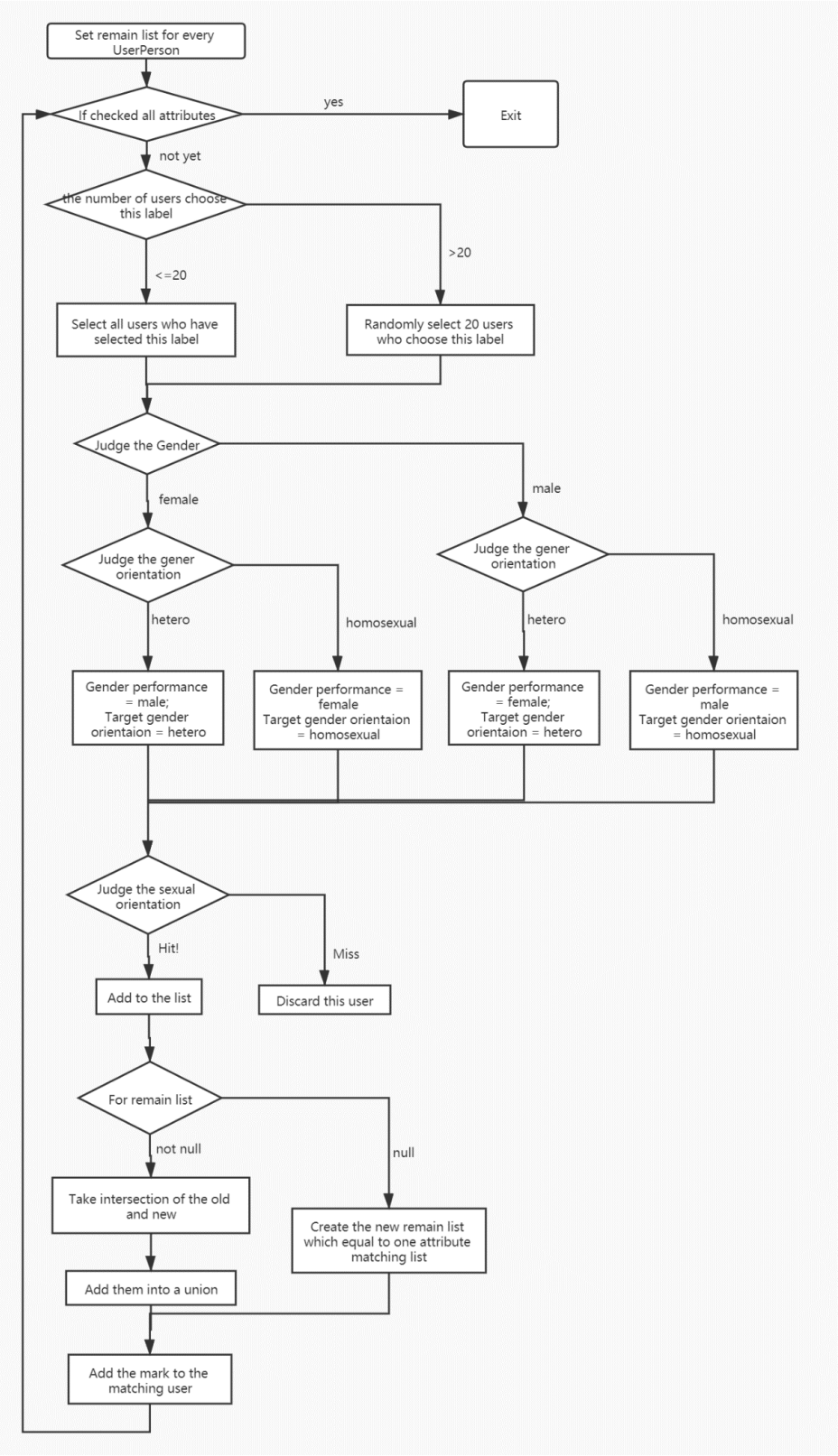
```java
package database.supports;
import ...

public class JDBCTool {

    public static Connection getConnection(String url, String dbname, String username, String password) {
        Connection conn = null;
        try {
            Class.forName("com.mysql.cj.jdbc.Driver");
            conn = DriverManager.getConnection( url: "jdbc:mysql://"+url+"/"+dbname+"?serverTimezone=Asia/Shanghai", username, password);
        } catch (SQLException | ClassNotFoundException e) {
            e.printStackTrace();
        }
        return conn;
    }

    //TODO Question 1 Check user name and password
    public static Connection getConnection() {
        return JDBCTool.getConnection( url: "localhost:3306", dbname: "groupwork", username: "root", password: "premacybenny");
    }
}
```

We use fuzzy search through partial letter of information by users, find out the object closest to its search content in the table, and list some objects that may be in line with the user's search. The SQL statement for fuzzy search is LIKE instead of '='. The wildcard character '%' could replace the missing letters. Add 'order by replace(,'',')' function at end can sort the result by the degree of proximity of the selected content, rather than by the original order of the primary. 'limit' statement is used to limit the length of the result, maximum show the first 20 result object.

```java
public UserPerson selectUserPersonByMentorID(int mentorID) {
    try {
        String sql = "SELECT * FROM UserPerson WHERE WHERE MentorID like " + mentorID + "% order by replace(MentorID," + mentorID + ",'') limit 20;";
        PreparedStatement ps = connection.prepareStatement(sql);
        System.out.println(ps);
        ResultSet rs = ps.executeQuery();
        rs.next();

        int systemID = rs.getInt( columnLabel: "SystemID");
        int userID = rs.getInt( columnLabel: "UserID");
        String emailAddress = rs.getString( columnLabel: "Emailaddress");
        String wechat = rs.getString( columnLabel: "Wechat");
        String genderOrientation = rs.getString( columnLabel: "GenderOrientation");
        Date dataOfBirth = rs.getDate( columnLabel: "DataOfBirth");
        String slogan = rs.getString( columnLabel: "Slogan");
        int work = rs.getInt( columnLabel: "Work");
        String surname = rs.getString( columnLabel: "Surname");
        String forename = rs.getString( columnLabel: "Forename");
        String gender = rs.getString( columnLabel: "Gender");
        String screenName = rs.getString( columnLabel: "ScreenName");
        String headIcon = rs.getString( columnLabel: "HeadIcon");
        String password = rs.getString( columnLabel: "password");

        ps.close();
        UserPerson userPerson = new UserPerson(systemID, userID, emailAddress, wechat, genderOrientation, dataOfBirth,
                slogan, work, mentorID, surname, forename, gender, screenName, headIcon, password);
        return userPerson;
    } catch (SQLException e) {
        e.printStackTrace();
        return null;
    }
```

Web severs

Recommendation



For recommendation, which is the most complex logic part of this project, the process is shown in the flow chart. Set one list called remain, create a loop for every attribute to compare and add mark for the object. For every chosen label, check the number of users choose it. If it > 20, we only select 20 of the people randomly, if it < 20, we select all the people selected this label, and check their gender. For this selected list, we determine their sexual orientation. According to the user's gender and its target gender, we only do the next operations for the coincident ones.

For remain list, if it is new and no object in the list, we set the selected list as the remain list. If it is not, united the newly selected people with the remained people from last time, and add the mark to them. Finally, union the people to join the new parts. Since the initial mark of each is 1, we do not add extra points to new parts.

We did not use any additional complexity to compare scores. We used the compareTo function and compareNode, which we talked about in our object-oriented programming class.

With this method, the recommendation has strong randomicity, strive for each recommended candidate is different, and have some common interests. It's more practical.

Web pages

After learning the lecture7:CREATE YOUR FIRST INFORMATION SYSTEM, we start to design the Web part.We know that HTML is a language used to describe web pages: Hyper Text Markup Language. Not like other language we learned before, HTML is not a programming language, but a markup language which is a set of markup tags. HTML uses markup tags to describe Web pages.
When create a HTML file, we will see this :

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>

</body>
</html>
```

<!DOCTYPE html> The declaration is at the beginning of the document, it used to tell the Web browser which HTML version the page is using. Here we use HTML5.
<html lang="en"> It means the web page uses English language.
<head> It is a container for all header elements.
<meta charset="UTF-8"> It tells the browser what character encoding format the page belongs to and then the browser can do translating job.
<title> It defines the title of a document and places it in the title bar or status bar of the browser

window.

To learn the more knowledge of HTML, I use *www.bilibili.com to search some videos, here is two main courses I learned:*

https://www.bilibili.com/video/BV1qv4y1o79t?p=1

https://www.bilibili.com/video/BV1uy4y1s7yr?p=1

At the same time, I use *www.runoob.com/html* to learn how to create website.

1. Login page

The function of this page is to allow login process for all the persons uses their screen name (unique) and password. We need two input bars which are user name and password, we also need two buttons which are login and register. If the user name and password is correct, then the user can login, if not, then there will be response: "username does not exist" or "wrong password".

To implementation the function, first we use <h1> to set the headline of the page, eg.

```
<h1>LOGIN</h1>
```

Then we use <form> to create an HTML form for user input, use <input> to create a input field to enter user name and password, use <button> to define "login" and "register" buttons to submit the data. Furthermore, we want to make them to be center of page, so we use to put them into a block and add attribute of center.

```
<div style="text-align: center">
  <form>
      <input type = "text" placeholder="NAME" name = "username" > <br>
      <input type = "password" placeholder="PASSWORD" name = "password"> <br>
      <button type="button">login</button>
      <button type="button">register</button>
  </form>
</div>
```

The preliminary page looks like:

2. Users' registration page

The function of this page is to add user's information in PERSON table. The users should enter the username, password, surname, forename and choose their gender (male or female).

To achieve these function, we set four input bars and a button to submit the user's information. We use <table> to distribute them, <tr> defines the row, <td> defines the table cell. We also use <input type="radio"> to create two radios (male and female). If the user did not input appropriate username, there will be response: "username already exist, please change one!", "cannot enter empty string!"

Here is main part of code:

```
<table align="center" >
  <tr>
    <td><label for="username">username</label></td>
    <td><input type="text" name="username" id="username" placeholder=" input your
            username"></td>
  </tr>
<tr>
  <td><label> gender</label></td>
  <td>
    <input type="radio" name="sex" value="male">Male<br>
    <input type="radio" name="sex" value="female">Female
  </td>
</tr>
```

The preliminary page looks like:

3.Users' attributes page

This page can set the user's screenname, E-mail, Wechat, birthday, gender orientation, slogan, sports, foods, travel footprint and music.

During studying, we know that we can define numerous input type, if we want to define email input part, we can use <input type="email">. In the same way, we can use <input type="date"> to define birthday part.

```
<td><input type="date" name="birthday" id="birthday" ></td>
```

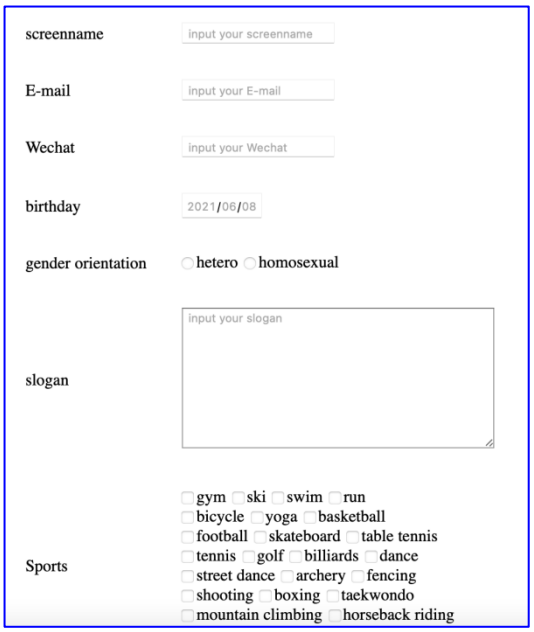
In addition, when user input slogan, the sentence may be very long, so we can use <textarea type="text" rows="xxx" cols="xxx"> to set the input area.

```
<td><textarea type="text" rows="10" cols="40" name="slogan" id="slogan" placeholder=" input your
slogan"></textarea></td>
```

When users choose their favorite sports, foods, music and their travel footprint, they can make the
multiple choice, so we use <input type="checkbox">.

```
<td><label> Sports</label></td>
<td>
  <input type="checkbox" name="sports" value="gym">gym
```

The preliminary page looks like:



4. Upload head icon page

This page can upload user's head icon.

We use <img> to define the image and set the size of the image.

```
<img   width="400" height="400">
```

We also need two input bars to select file and submit.

```
<button type="file" name="photo" size="50" >Choose a picture!</button>
<button type="submit" >Save</button>
```

The preliminary page looks like:

5.User's main page

This page is user's main page, the users can change their characters, head icon and information. They can also look up system recommendations, the events which will be hold, they like whom or who likes them and their mentors.

Here we use <a> for these buttons which defines a hyperlink that is used to link from one page to another. In this step, we began to use Bootstrap which is a quick approach to page layout. We mainly reference *https://v3.bootcss.com*.

Bootstrap is by far the most popular front-end framework. Bootstrap is based on HTML, CSS and JavaScript, which is simple and flexible and makes Web development faster.

We can link resource file of bootstrap to our project.

```
<link rel="stylesheet" type="text/css" href="css/bootstrap.min.css">
```

href="login.html" means when user clicks the “log out”, then it will jump to the login.html.

Class="btn btn-info" means adding color to the <a>.

```
<a class="btn btn-info" href="login.html">log out</a>
```

There are different color style in Bootstrap:

EXAMPLE

（默认样式）Default （首选项）Primary （成功）Success （一般信息）Info （警告）Warning

（危险）Danger （链接）Link

Copy

```
<!-- Standard button -->
<button type="button" class="btn btn-default">（默认样式）Default</button>

<!-- Provides extra visual weight and identifies the primary action in a set of buttons -->
<button type="button" class="btn btn-primary">（首选项）Primary</button>

<!-- Indicates a successful or positive action -->
<button type="button" class="btn btn-success">（成功）Success</button>

<!-- Contextual button for informational alert messages -->
<button type="button" class="btn btn-info">（一般信息）Info</button>

<!-- Indicates caution should be taken with this action -->
<button type="button" class="btn btn-warning">（警告）Warning</button>

<!-- Indicates a dangerous or potentially negative action -->
<button type="button" class="btn btn-danger">（危险）Danger</button>

<!-- Deemphasize a button by making it look like a link while maintaining button behavior -->
<button type="button" class="btn btn-link">（链接）Link</button>
```

For the other part, we can just use to show them.

```
<span class="label label-info">screenname</span>handsomeboy
```

The preliminary page looks like:

However, we found the layout is not good. Then we decided to combine the change part. At first, we wanted to set a navigation bar like :

However, when we tried to combine the change my characters, change my head icon, change my information into one Change, we needed to use JavaScript, it is not easy to implement. We decided to use another way which is keeping the original button and adding drop-down menu. This step we learned from *https://v3.bootcss.com/components/#dropdowns*

```
<div class="dropdown">
    <button class="btn btn-toolbar">change</button>
    <div class="dropdown-content">
        <a class="btn btn-info" href="Users_Attributes.html">change my characters</a>
        <a class="btn btn-info" href="head_icon.html">change my head icon</a>
        <a class="btn btn-info" href="modify.html">change my information</a>
    </div>
</div>
```

We need locate the drop-down content. The position attribute specifies the type of the positioning method for an element. With absolute positioning, an element can be placed anywhere on the page. Inline-block makes an element an inline element, having the property of an inline element that it can share a line with other inline elements, but does not monopolize a line.

```
.dropdown {
    position: relative;
    display: inline-block;
}
```

We need hide the drop-down content by default.

```
.dropdown-content {
    display: none;
    position: absolute;
    background-color: #f9f9f9;
    min-width: 160px;
    box-shadow: 0px 8px 16px 0px rgba(0,0,0,0.2);
}
```

We also set the menu style with:

```
.dropdown-content a {
    color: black;
    padding: 12px 16px;
    text-decoration: none;
    display: block;
}
```

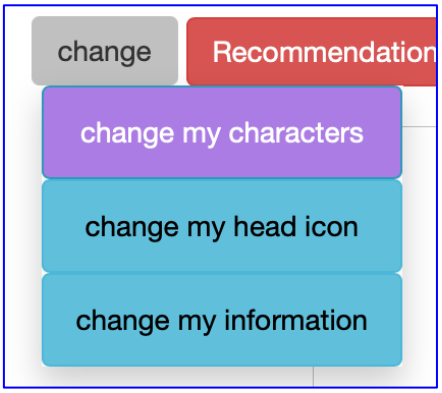
When the mouse move onto the change button, it will pop the menu.

```
.dropdown:hover .dropdown-content {
    display: block;
}
```

If the mouse move onto one of the choice, the color will be change.

```
.dropdown-content a:hover {background-color: #ac79e6}
```

Now the button looks like this :



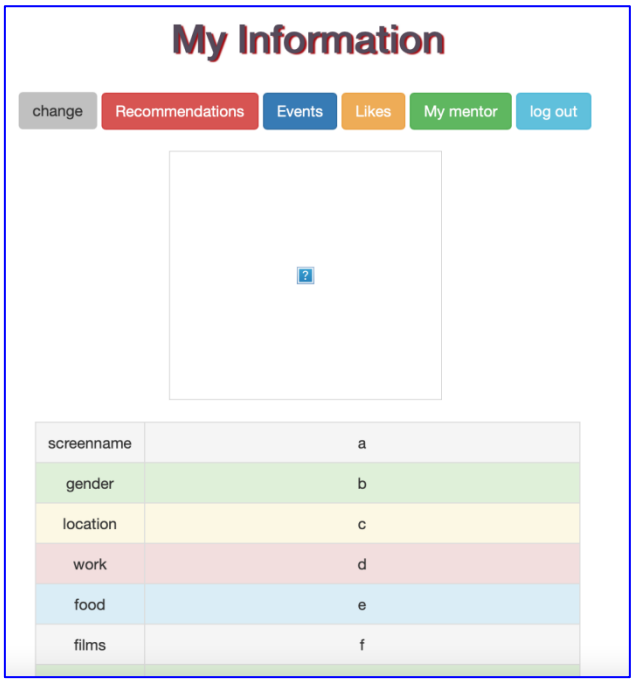
To make other part more comfortable, we use table to lay out and set different color on each line.

. table-Bordered class adds borders to the table and each cell in it.

.table-hover class causes each row in the to respond to the mouse hover state.

```
<table   class="table table-hover table-bordered " >
    <tr class="active">
        <td align="center" width="100px">screenname</td>
        <td align="center">a</td>
    </tr>
```

Now the page looks like:



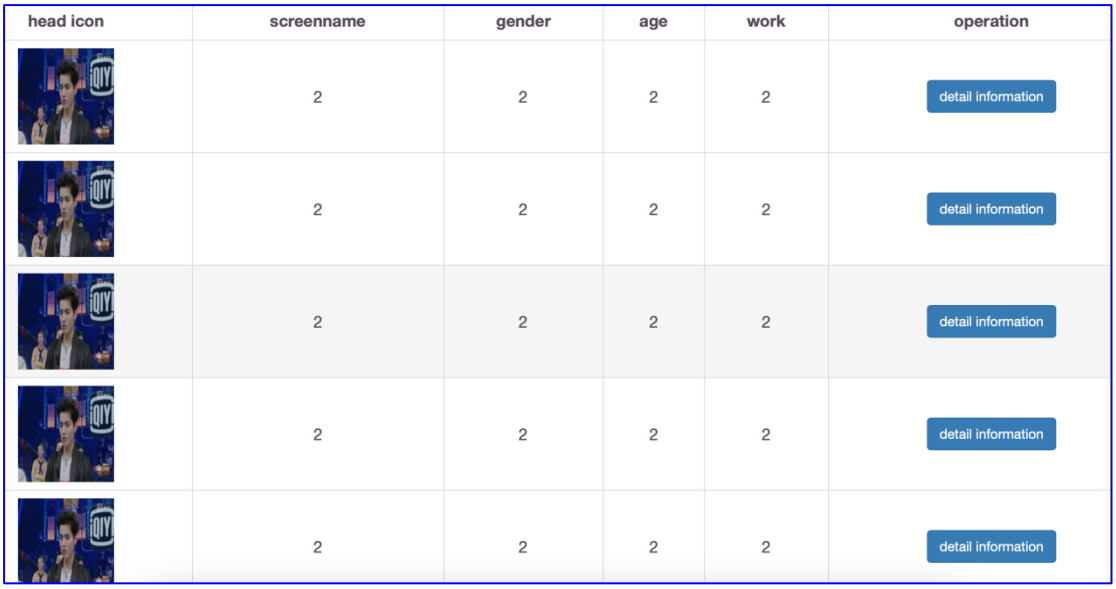
6.Recommendations

In this page system recommends the matching users. The user can see their head icon, screenname, gender, age, work. If the user is interested in someone, then the user can look up him/her detail information.

To add the head icon into the table , we use <img> and define the size of image.

```
<td align="center" ><img src="image.jpg" width="100" height="100"></td>
```

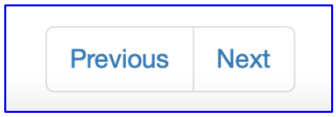
The page looks like:



Furthermore, we set one page can only display five matching users, so we need paging buttons. class="pagination pagination-lg" is from Bootstrap which can make size of the button bigger.

```
<nav aria-label="Page navigation">
  <ul class="pagination pagination-lg">
<li>
      <a href="#" aria-label="Previous">
        <span aria-hidden="true">Previous</span></a> </li>
<li>
      <a href="#" aria-label="Next">
        <span aria-hidden="true">Next</span></a></li></ul></nav>
```

The paging buttons looks like:



7.Events

In this page users can choose the events they want to join. Each event shows the location type, geographical location, activity time, activity. We put them into a table and add operation button to let users make choice.

The preliminary page looks like:

Location type	Geographical location	Activity time	Activity type	Operation
1	2	3	4	quit
1	2	3	4	join
1	2	3	4	join
1	2	3	4	join
1	2	3	4	join
1	2	3	4	join
1	2	3	4	join
1	2	3	4	join
1	2	3	4	join
1	2	3	4	join

8.Likes

The users can look up the people who are interested in them and whom they have been interested in. This page is same like Recommendations Page, which shows users' head icon, screenname, gender, age, work and some operations such as seeing detail information, upgrading relations, degrading relations, canceling relations.

To make the content in the table be center horizontally and vertically, we use:

```
<td class="text-center"  style="display:table-cell; vertical-align:middle;"> 2</td>
```

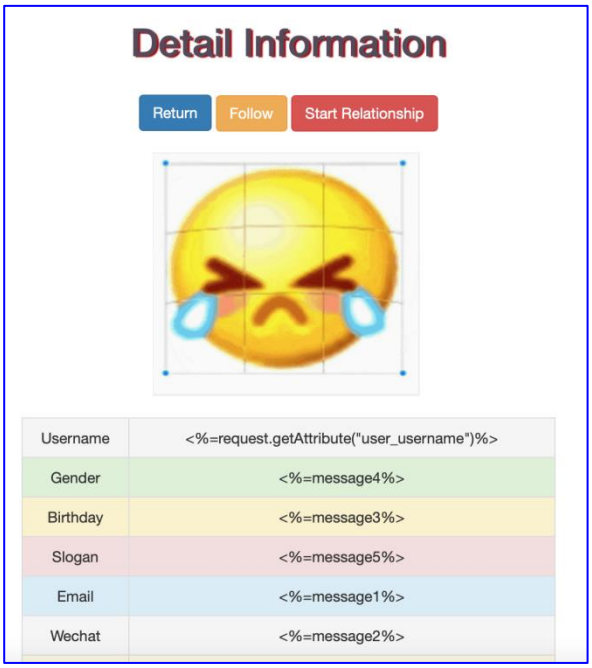
The preliminary page looks like:

head icon	screenname	gender	age	work	operation
	2	2	2	2	detail information Upgrade relations Degrade relations Cancle relations
	2	2	2	2	detail information Upgrade relations Degrade relations Cancle relations
	2	2	2	2	detail information Upgrade relations Degrade relations Cancle relations
	2	2	2	2	detail information Upgrade relations Degrade relations Cancle relations

9. Detail information

This page is same like User's main page, but the function has a little change. The users can follow the people if they are interested in or start relationship.

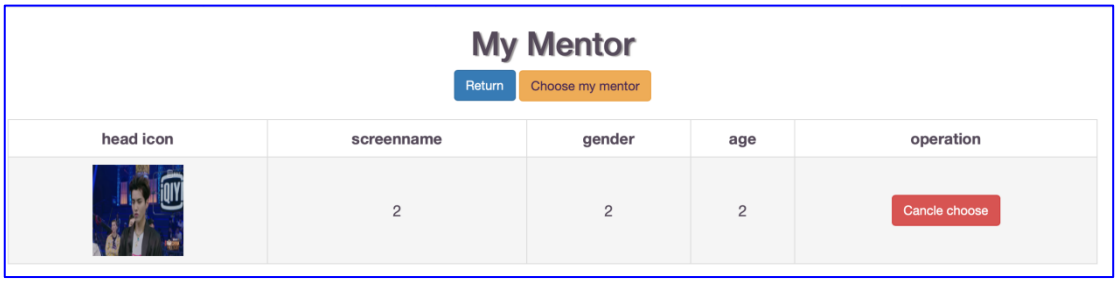
The preliminary page looks like:



10.My mentor

This page users can see the mentor they have chose, and they can cancel their choose. If they have not chose the mentor, they can click “choose my mentor” to make choice.

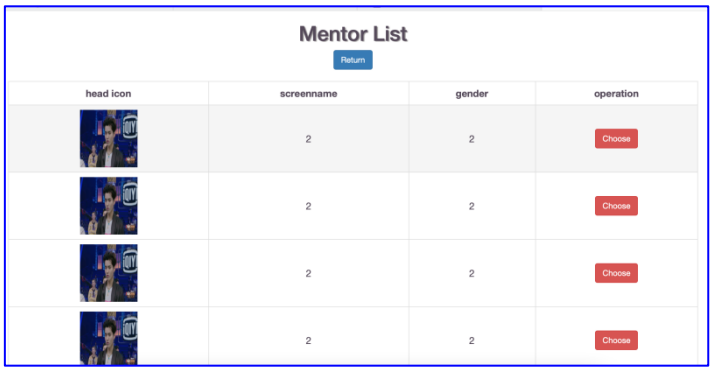
The preliminary page looks like:



11.Mentor list

As we talk before, users can choose their mentors and there will be five mentors to let them choose.

The preliminary page looks like:



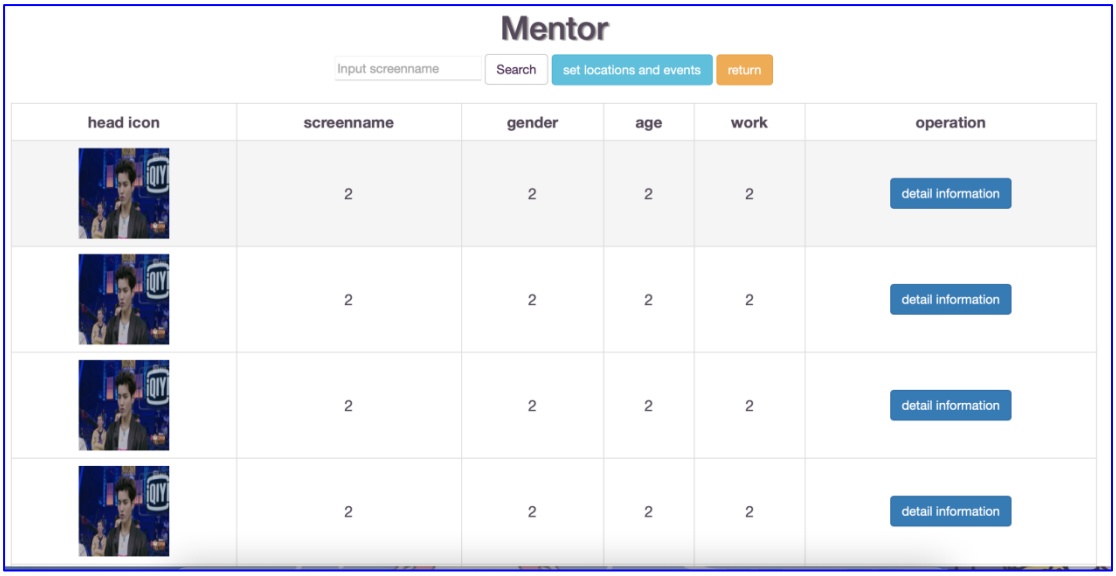
12.Mentor

The mentor can see all of the users and search the users by screenname. In addition, the mentor can set locations and events.

To realize the search function, we use

```
<input   type="text" name="find_screenname" id="find_screenname" placeholder="Input screenname">
<button type="submit" class="btn btn-default" style="color: rgb(73, 41, 85)">Search</button>
```

The preliminary page looks like:



13.Location_Event

The mentors can add events in this page, they need set location type, geographical location, activity time, activity type.

It is worth mentioning that we use <select> to define the location type.

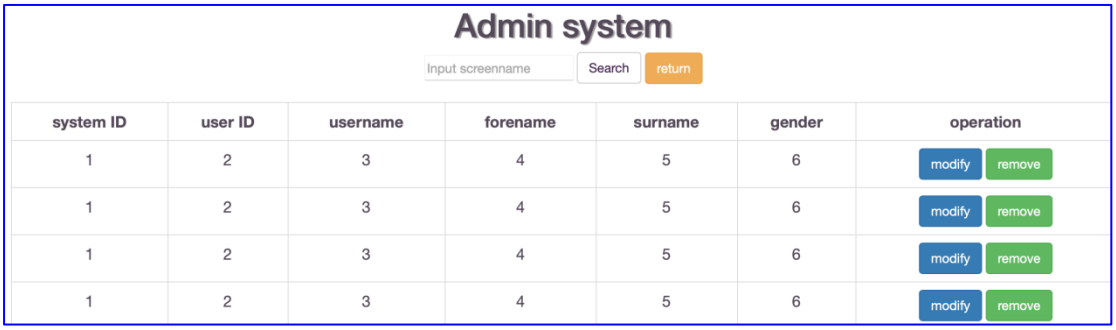
```
<select class="form-control "    style=" width: 200px;">
    <option>coffee/tea house</option>
    <option>bar</option>
    <option>restaurant</option>
    <option>shooping centre</option>
    <option>park</option>
    <option>cinema</option>
    <option>theatre</option>
</select>
```

We put the setting menu into the table and the mentors can add or remove the decision they have confirmed.

The plan of page is like:

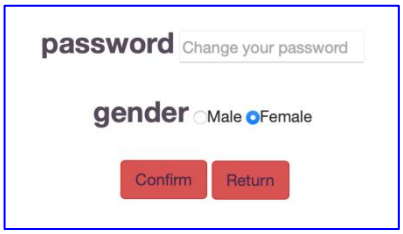
14.Admin

This page is same with Mentor page, the admin can look up all users and modify their information.



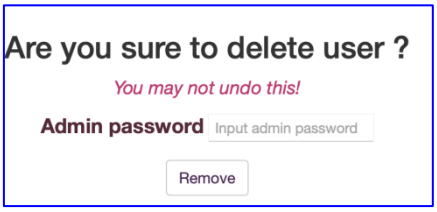
15.Modify

This page is same with Register page, the admin can change user's basic information.



16.Remove

Admin can remove the users and the admin need to input admin password to do it.



In terms of beautifying pages with CSS, we take blue and purple as the main colors, which can both reflect the romantic feeling of our system theme, but also distinguish it from other dating matching interfaces with too gorgeous and conventional systems.

To use CSS better, we search https: //www.runoob.com/css/css-tutorial.html and also use bookstrap to finish it.

1. **Login page**

We hope to use ripples to match body colors in login page and also add a touch of interest to our page.

First, we implement the corrugated icon with the *'svg'* graphics. Because the ripple changing is a gradient-like pattern, we let it accept a *'viewbox'* property that makes it an adaptive size scale rendering in any window, setting both the size and scale to fit the adaptive window size. Then, we then use 'defs' and 'path' to determine the shape container size of the corrugated graphics. Next, we combine the graphics by *'g'* and achieve stacked spatial relationships between the ripples: changing the coordinate values of each ripple.

```
<svg class="waves" viewBox="0 24 150 28"
  preserveAspectRatio="none" shape-rendering="auto">

  <defs>
    <path id="gentle-wave"
          d="M-160 44c30 0 58-18 88-18s 58 18 88 18
          58-18 88-18 58 18 88 18 v44h-352z" />
  </defs>

  <g class="parallax">
    <use xlink:href="#gentle-wave" x="48" y="0"
    fill="rgba(255,255,255,0.7)" />
    <use xlink:href="#gentle-wave" x="48" y="3"
    fill="rgba(255,255,255,0.5)" />
    <use xlink:href="#gentle-wave" x="48" y="5"
    fill="rgba(255,255,255,0.3)" />
    <use xlink:href="#gentle-wave" x="48" y="7"
    fill="#fff" />
  </g>
</svg>
```

We also want to achieve the background color gradient in *style.css*, which requires the *'linear-gradient'*. So we set the gradient axis to 60 degrees, starting from purple, to the excessive to blue gradient after purple gradient completes.

```
.header{
    position: relative;
    text-align: center;
    background: linear-gradient(60deg, rgb(84,58,183,1)0%, rgb(0,172,193,1)100%);
    color: white;
    padding-top: 10vh;
}
```

For the title and corrugated layout we use the *'flex'* property to give the container maximum flexibility and place it in center.

```
.lnner-header{
    height: 20vh;
    width:100%;
    padding:0;
    /*flex*/
    display:flex;
    justify-content: center;
    align-items: center;
}
```

And for the establishment of the corrugated shape, the focus is on the highest and lowest points of the ripple floating, implemented by *'max-height'* and *'min-height'*.

```
.waves{
    position:relative;
    width:100%;
    height:25vh;
    margin-bottom: -7px;
    /*minnimum*/
    min-height:130px;
    /*maxmum*/
    max-height:25vh;
}
```

Then, in order to enable the ripple animation effect, we use *'@keyframes'*, 0% positioning the beginning of the fluctuation, 100% positioning fluctuation end.

```
/*animation*/
@keyframes move-forever{
    0%{
        transform: translate3d(-90px,0,0);
    }
    100%{
        transform: translate3d(85px,0,0);
    }
}
```

Next, we use *'parallax'* to achieve the dynamic change and varying time of ripples between the start and end positions.

```
.parallax>use{
    animation: move-forever 25s cubic-bezier(.55,.5,.45,.5)infinite;
}
```

For each ripple, the changes are different, we use *'animation-delay'* to achieve the time difference of fluctuations between each ripple and apply different ripples with *'nth-child'*.

```
.parallax>use:nth-child(1){
    /*delay:2s*/
    animation-delay: -2s;
    /*overall time: 7s*/
    animation-duration: 7s;
}
.parallax>use:nth-child(2){
    animation-delay: -3s;
    animation-duration: 10s
}
.parallax>use:nth-child(3){
    animation-delay: -4s;
    animation-duration: 13s;
}
.parallax>use:nth-child(4){
    animation-delay: -5s;
    animation-duration:20s;
}
```

Finally, we also set the fluctuation of how to perform when the screen is less than 768px (namely the phone side) to meet more needs.

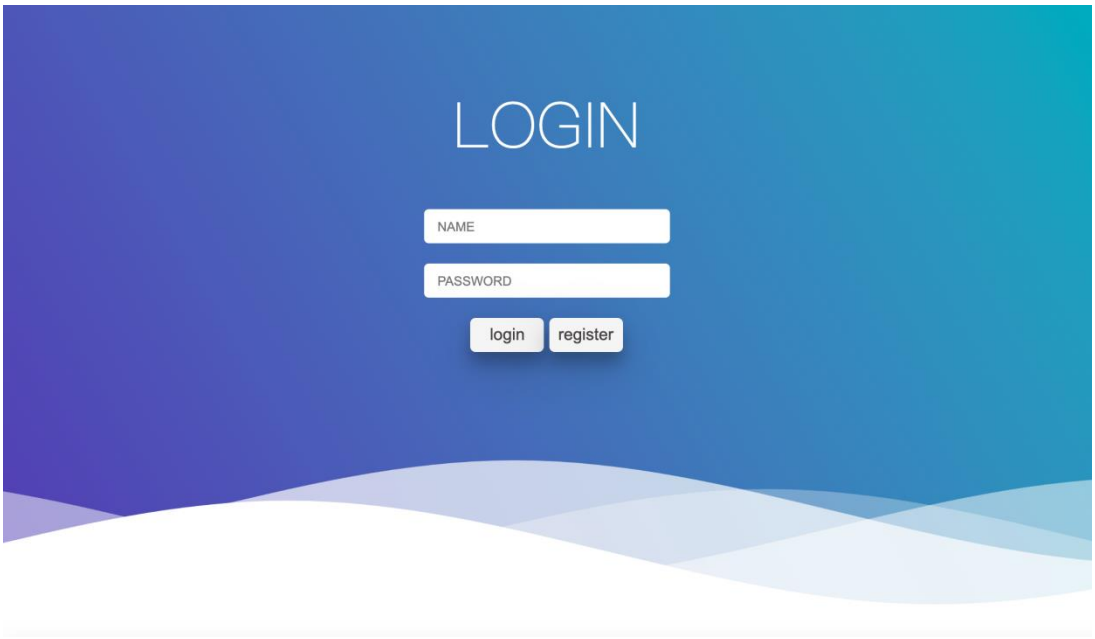
```
/*<768px*/
@media(max-width:768px){
    .waves{
        height:40px;
        min-height: 40px;
    }
    h1{
        font-size:24px;
    }
```

For other parts, such as buttons, we also made interesting changes. For example, the status of button during different operations: *after and active*, changes the color of the button when the mouse clicks the button. What's more, when we click the button, the button will also have a ripple effect.

```
button {
    position: relative;
    background-color: whitesmoke;
    border: none;
    font-size: 18px;
    color: #464242;
    padding: 8px;
    width: 80px;
    border-radius: 6px;
    text-align: center;
    -webkit-transition-duration: 0.4s; /* Safari */
    transition-duration: 0.4s;
    text-decoration: none;
    overflow: hidden;
    cursor: pointer;
    box-shadow: 0 12px 16px 0 rgba(0,0,0,0.24),0 17px 50px 0 rgba(0,0,0,0.19);
}
```

```
button:after {
    content: "";
    background: #90a6ee;
    display: block;
    position: absolute;
    padding-top: 300%;
    padding-left: 350%;
    margin-left: -20px!important;
    margin-top: -120%;
    opacity: 0;
    transition: all 0.8s
}

button:active:after {
    padding: 0;
    margin: 0;
    opacity: 1;
    transition: 0s
}
```

Here is the page after using CSS.



2. Register page

For the register page, we insert a picture to beautify the page. This is done by *'background-image'*.And we change the *'width'* of the information bar by inserting pictures to the right side of the border.

```
.login_ad{
    width: 700px;
    height: 600px;;
    overflow: hidden;
    background-image: url('picture.gif');
    background-size: cover;
    background-repeat: no-repeat;
    background-position: center center;
}
```

To make the size of the message bar and image more appropriate, we change the distance and color of the message bar from the bottom border by calling *'l_float'*.

```
.l_float {
    float: left;
    height: 600px;
    background: white;
}
```

We use clear to specify that information bar elements are not allowed to float.

```
.l_clear {
    clear: left;
    display: block;
    overflow: hidden;
    content:"";
}
```

To make the image clear and comprehensive, we call *'login_body'* and the other classes to determine the size of the box and the height of the page.

```
.login_body{
    width: 1000px;
    margin: 0 auto;
    margin-top: 80px;
}
.container.login_body.login_form{
    width: 300px;
}

.login_top,.login_con{
    background-color: #fff;
}

.login_top{
    padding: 20px 30px 10px 30px;
    font-weight: 900;
    font-size: 21px;
}
```

To adjust the position of the input box in the message bar, we call login_con to change its perimeter distance.

```
.login_con{
    padding: 20px 30px;
}

.login_con div{
    padding-bottom: 20px;
    position: relative;
}


.login_con>form>div>input{
    display: block;
    margin-top: 10px;
}
```

To avoid having too many elements make the page too clutter, we remove the border of the page information bar through *'outline'* and underscore through *'border-bottom'*.

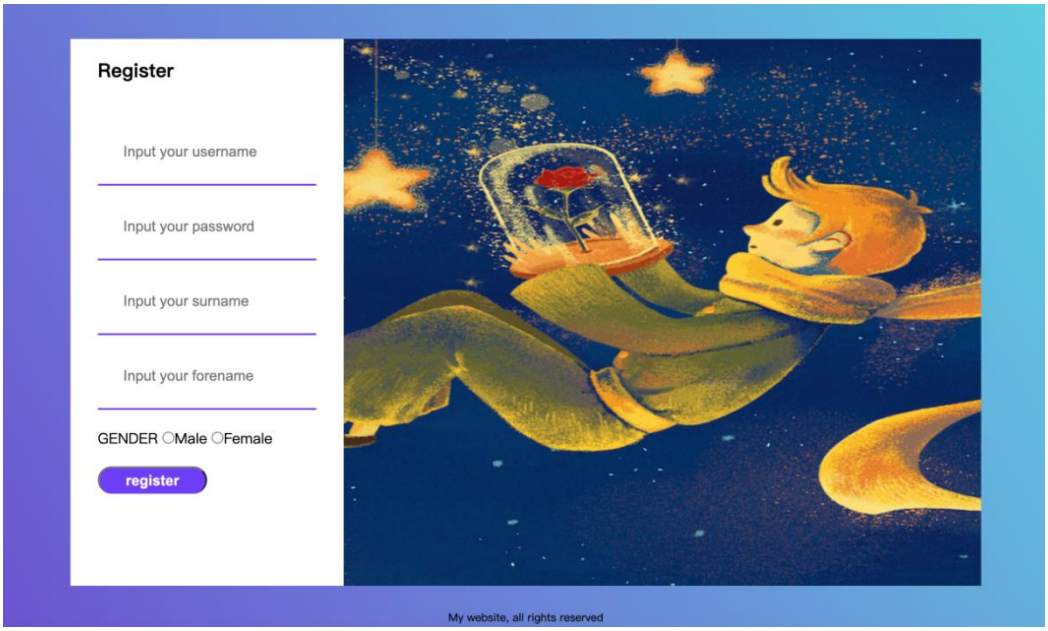
```
.login_con>form>div>input{
    padding: 0 28px;
    outline: none;
    height: 70px;
    width: 184px;
    border: none;
    border-bottom: 2px solid #703EFF;
    color: #333;
    font-size: 16px;
}
```

To coordinate the button and the page overall, we have also changed the background color

of the button.

```
.login_con button{
    background-color: #703EFF;
```

Here is the page after using CSS.



3. User _Attributes page

For User_Attibutes pages, we want to increase the interest of users filling in more information by adding the color types of page background transformation, which is still achieved through *'linear-gradient'*. '*Animation'* is used to control the time of background color changing.

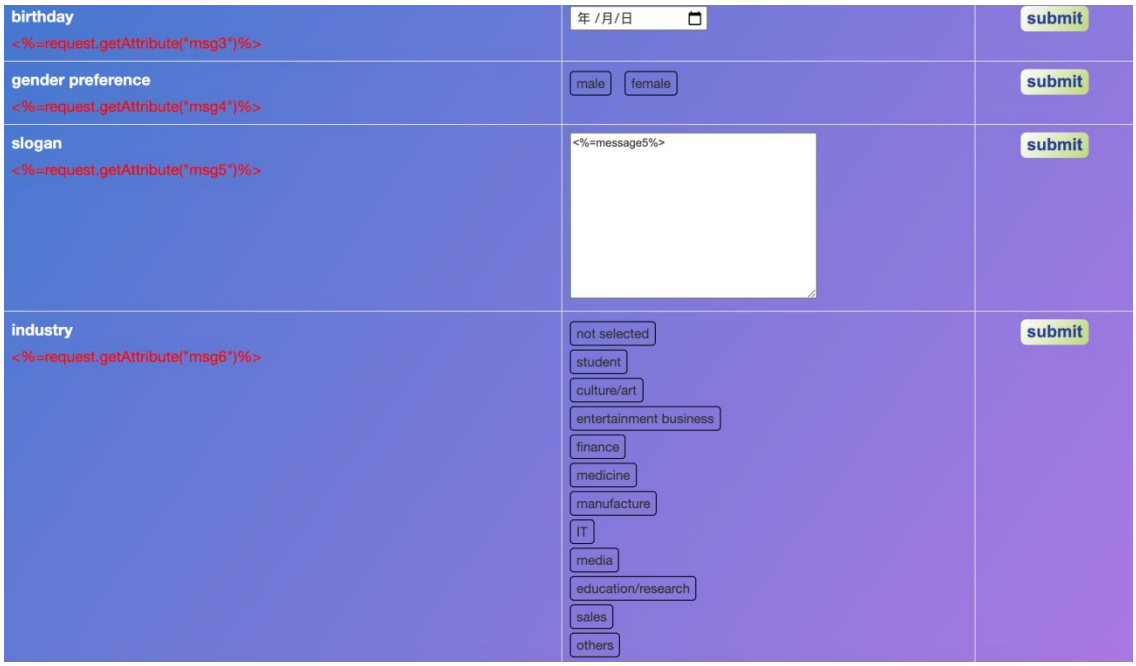
```
body{
    background-image: linear-gradient(125deg, #305f8d, #58d38b, #4979d3, #ac79e6, #da6bb8);
    background-size: 400%;
    animation: bganimation 15s infinite;
    max-width: 1200px;
    margin: 0 auto;
}
```

To make the page look more neat, we decide to click the options directly, not through the buttons. So we also give the '*class' of* options different statuses: *input and active*. This makes the color change after clicking the option, proving that the selection has been made. 'Opacity' is used to control the opacity of an element's text, where 1 is opaque and 0 is entirely transparent.

We also use *'margin'* and *'padding'* to adjust the distance around the button, '*border-radius'* to make the corners of the button rounded, '*border'* to change the color of the border, and *'line-height'* to change the position of the button.

```css
.input-wrapper{
    position: relative;
    margin: 0 10px 10PX 0;
    padding: 4PX 6PX;
    border-radius: 4PX;
    border: 1px solid #000;
    line-height: 30px;
}

.input-wrapper input{
    opacity: 0;
    position: absolute;
    top: 0;
    left: 0;
    width: 100%;
    height: 100%;
}

.input-wrapper.active{
    background: black;
    color: #FFFFFF;
}
```

Here is the page after using CSS.



4. User Pages, Head_icon Pages, Event pages, Likes pages, Mentor Pages, My_mentor pages, MentorList Pages, Add pages, Remove pages, Admin Pages, Modify Pages, Recommendations pages

For these pages, as a branch of the main page, we still set the background color to a blue gradient.

```css
body{
    background-image: linear-gradient(125deg, #a1e2dd, #627db6);
    background-size: 400%;
    animation: bganimation 15s infinite;
}
```

For some pages like remove which has buttons in it. We also set the type with suitable colors.

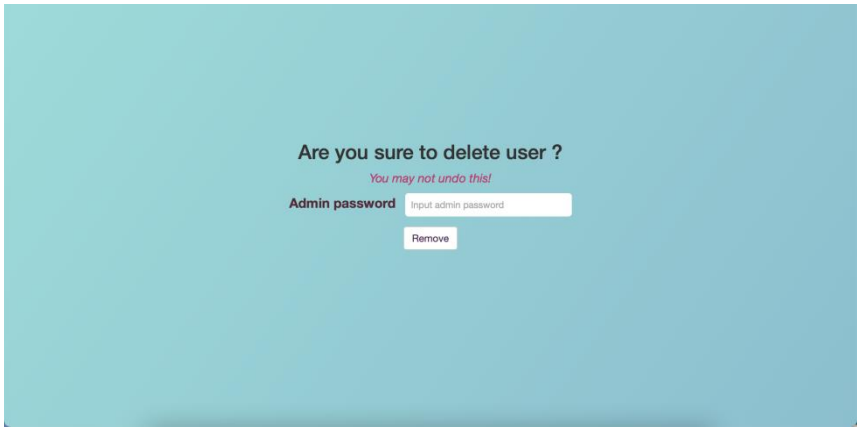
```
<a class="btn btn-default" style="color: rgb(73, 41, 85);font-size: 15px;" href="#" role="button">Remove</a>
```

For some pages like events which has table, to make our page look better, we adjusted the spacing, height with the top, and center the table.

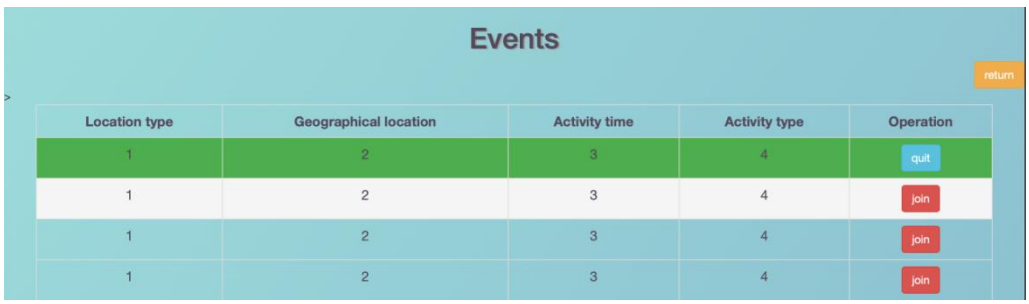
```css
tr{
    height: 30px;
}

td.text-center{
    line-height: 30px!important;
    font-size: 17px;
    color: rgb(85, 74, 90);
}
```

Here are the pages after using CSS.

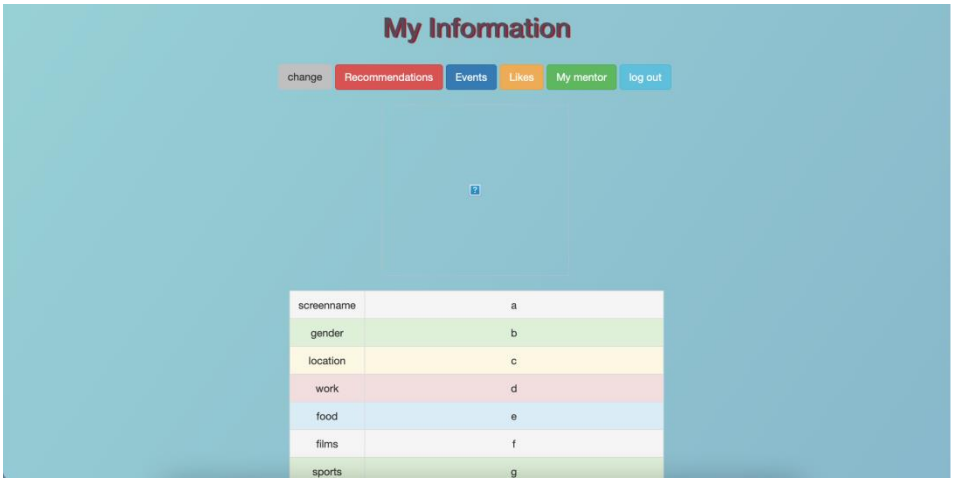
Remove Page:



Events Page:



User Page:



Team Member Contribution.

Yushi Wang and Benteng Ma designed the architecture of this system and the ER diagram of this project.

Mengxi Zhao and Fengbo Zhang mapped the tables from the ER diagram and do normalization of every table.

Yushi Wang establish every table and wrote DAOs and objects of every table.

Benteng Ma wrote the servlets files and connected the database and web pages.

Mengxi Zhao, Fengbo Zhang and Benteng Ma design the function of pages and Mengxi Zhao and Fengbo Zhang implemented them with jsp and css.

Self-assessing and self-feedback records.

What we've learnt:

We have a very clear understanding of database design and use. **In this project, the four of us did all the work from the determination to the completion of the project.** From the determination of the project theme to the design of the ER diagram, Mapping. Normalization of tables and its establishment of a complete database. The writing of back-end functions and the connection of front and back ends using servlets. Almost everything can be completely mastered by the completion of the project. What's more, we become more familiar with object-oriented programming.

About the web page, we deepened our understanding of **HTML** and **CSS**, tried to change the background of the page, adjusted the format of the buttons, and got familiar with how to arrange the layout of the page, and how to adjust the paragraphs and fonts of the page. With a good label means to build a good house, and then you can use CSS to beautify the house, make the house look better, front-end design I think to be familiar with all kinds of tags, in order to better layout of the web page, more easily to complete the task. We got familiar with various CSS selectors, improved our ability to use CSS, and learned how to use Bootstrap plugin to beautify our pages in a more convenient way.

Besides, we have enriched the experience of team cooperation, how to reasonably allocate tasks in the process of cooperative development and finish tasks in time.

Our regrets in this project:

Since we had never done such a development project before, we had some problems in terms of timing and how hard we expected the project to be. The lack of experience in the design led to over-optimism. Although there was a relatively complete presentation, the goal was not fully reached.

When designing the database, the difficulty of the future implementation project was not taken into account, and the difficulty of the overall control was not enough. As a result, we left three parts unfinished: Date/Relationship/user photo.

User photo: It took us too long to upload our head icon, but it still didn't work well. We think this is Tomcat's problem. All in all, the lack of time later led to no more energy to revise this part. Our lack of understanding of Tomcat is the main reason.

Relationship, Date: It is just like the "like" page. Relationship is a one-to-one relationship that requires confirmation by both parties. We designed it as a relationship, but when we implemented it, we found that we hadn't thought through its interaction logic. We didn't finish it because of time.

The structure of the code is not concise enough. There's a lot of overlap that can be packeting. We found this flaw during development, and it caused our development process to be too

difficult. This directly causes the program to be too large and slow to run. If we have time we will clean up the code more succinctly.

When designing a database, many functions are designed but not used. During development we also found that many functions were added later, which led to duplication of design and slow development.

We don't have any experience in JSP and Web development. As a result, we have to do the project while learning. This also leads to a serious lack of communication in our work. The team members did not learn the same parts, so there was a big problem with the front and back connection parts. We realized that, the same web pages need more people to complete, even many of the unnecessary process is repetitive.

About the website, due to time constraints, we only learned to use **HTML** and **CSS** to design the web page and applied Bootstrap interludes with it. Unfortunately. We didn't use Java Script, and all the pages were **static**, which still had a certain gap with the current popular web page style. Besides, for technical reasons, there are still many deficiencies in the layout and beautification of some web pages. There is also some problems about the specific use of margin and border of the page. In the process of beautifying the page, we wanted to use a drop-down navigation bar, but due to formatting and other reasons, we didn't try it. Also, on some pages, we tried to use some borders, but it didn't work out because of browser differences, and this will be our next effort.

The difficulties and the solutions

1. Jumps between webpages and parameter sending

Difficulty: there are too different ways to jump: <a>, <form, post/get>… But they use different method to send information, some use request parameters, some use request attributes, some just don't. This brings difficulty for servlets.

Solution: we used several try… catch… s to make sure that all the information can be found, through parameters, request attributes, or session attributes. We also keep our mind to send them, and organize our parameters well.

Date 6.5-6.8

2. Hibernate do not support JOIN method

Difficulty: This is our original idea, call JOIN method in ORM. Hibernate do not support JOIN method in SQL, cannot print the union table of User and Person. However, this is our old idea, now we use JOIN and VIEW together in MySQL and use UserPerson object directly.

Solution: Called Both JDBC's Connection method and Hibernate ORM method. User the two options when necessary.

Date: 6.4

3. show information selectively

Difficulty, the JSP condition <c:> is very hard to use, especially with java, at least for us.

Solution, we settled our webpage to be static, and use <%= (condition) ? A : B %> to output our information, and even choose the target to jump to.

Date 6.7

4. The problem of show photos with tomcat

Difficulty: A locally saved user's head icon cannot be sent to the web page immediately via Tomcat and takes a long time to refresh.

Solution: It is the drawback of tomcat, and we cannot solve it totally. A good compromise is that forcing Tomcat to refresh by changing the local name of the photo uploaded by the user each time. The result was acceptable.

Date: 6.7

5. The name of photo

Difficulty: With the correct statement, logic and path of photo, Tomcat said it cannot find the photo that should already exist by the correct path. Sometimes it succeeds, but most of the time it fails.

Solution: Finally, we find the difference between the successful condition and the failed condition is that the name of photo. Photo titles must be in all lowercase letters, with no underlining or uppercase letters. It's a ridiculous problem, even there's explanation about it on the Internet. We can only change all the names.

Date: 6.6 we find the problem of name with underlining

6.8 we find the problem of name with uppercase letters

6. The recommended candidate only outputs the first person repeatedly

Difficulty: When we want to get mutiple object though one judgement condition. e.g. find all users' ID which interested sport's ID = 5. It will return a lot of IDs but is repetitive. We checked a lot of time, and the grammar and logic is correct.

Solution: After checking on the Internet, we know that if use multiple interfaces of Hibernate mapping, it will only display the first data queried repeatedly. After several fruitless modifications, we decided to go back to SQL method with Connection and Statement. And then we got a hopeful answer.

Date: 6.5

7. Hibernate delete object method

Difficulty: If the object being deleted has a null value in the table, the deletion will be recognized as an error and the deletion will fail.

Solution: Use JDBC Connection and Hibernate together to solve the problem

Date: 6.8

8. After Search

Difficulty: after search something, it's hard to cancel our search.

Solution: we restart the webpage, once click “cancel” button

Date: 6.10

9. A white edge appears at the top of the page

Difficulty：In the early stage of beautifying the login page, the top of the page will appear a white edge, affecting the overall beauty.

Solution: Use padding to adjust the distance at the top of the page.

Date: 6.5

10. Size of the background image

Difficulty：When you insert images in the background of the Register page, you will never be able to fill them up.

Solution：Set width in login_body and background_size of picture.

Date: 6.6

11. Adjust the frequency of the ripple at the bottom of the Login page

Difficulty：The ripples at the bottom of the Login page are always start at the same time and end at the same time。

Solution：Use animation_delay to change the time of each ripple.

Date: 6.8

Highlights of our system

1 Recommendation

Feature: Strong randomicity, strive for each recommended candidate is different, and have some common interests, which is more practical.

Realization method: See the flowchart for details.

Date: 6.4

2 Page Turning

Feature: allows user to view others page by page

Method: to keep the page number for each page, and let the iterator to iterate according to the pages, to push users.

Date: 6.8

3 upload and use personal head icon

Feature: allows user to have head icons, and upload their own head icons.

Method: use servlet stream to get file post from JSP, store the image at local directory, and save the path in the database.

Date: 6.7

4 jump from one webpage to several, with only one button

Feature: user can go back to different last pages though they just viewed the user_detail.jsp, this saves us a lot of work to make more webpages.

Method: use post method to send hidden values, so that the servlet can know the origin, and give the link to send the user back.

Date: 6.8

5 Usage of servlets

Feature: this allows to separate the connection between JSP and backend, and provides strong functions, with this in the middle, we can develop separately in parallel, which is very convenient.

Date: May - now

6 The use of Label Object interface

Feature: With the interface and standard form of interests, we could control the objects successfully. In every Std class, it could get every LabelID anf UserID, which is uniform and easy to manage.

Realization method: We created one interface LabelObject for all interests, like Sports, Books, Films, Food and traveled location and Stdxxxx for every interested which implement LabelObject interface and use Hibernate interface with the original table. These table have one common characteristic that they only include their label ID and user ID and they takes two together as their primary key. With this inerface, many method of different classes can be written into one.

Date: 6.3

7 The use of Hibernate

Features: Using Hibernate does not require calling a session every time, as opposed to using SQL statement methods that use Connection directly in JDBC. This saves a lot of time, especially when a large amount of data needs to be checked through a single table. Hibernate has a code reusability. Hibernate provides more write freedom for inserts and updates because Hibernate can return the created object directly.

Realization method: Inherits HibernateUtil for each class that could open session directly through HibernateUtil for later use and returns the desired object.

Date: Around 5.28

8 The use of Dynamic DAO

Features: The files in DynamicDAO are redesigned versions of the DAO and require an instance to manipulate the database. It is not static, and the session is closed only after all the required operations have been completed, which saves a lot of time for opening and closing.

Realization method: By inheriting DynamicDAO class, we unify the creation and closure of Session Connection and Statement SQL objects.

Date: 6.3

9 The compatibility of Hibernate and JDBC Connection

Feature: Because Hibernate has some disadvantages, such as cannot do multiple mapping, cannot JOIN the tables together. Compatible with both Hibernate and Connection methods and complete all desired functions with the maximum possibility of time saving.

Realization method: Both Hibernate queries and JDBC Connection SQL calls in DynamicDAO(Later deleted at 6.5) and LabelObject.

Date: 6.4

10 The use of prepared statement

Features: It is much more effective to prepare statements in advance. Prevent a malicious user who might attempt to enter data to alter the SQL.

Realization method: If there are parameters that can be changed, set placeholders so that they can be filled in later. Precompiled query into code that MySQL understands, so there is no need to do this every time the query is run.

Date: From MAY

11 The use of fuzzy search

Features: Through partial letter search of information, the user can find out the object that is closest to the content of the search in the table and list some objects that may be in line with the user's search.

Realization method: The SQL statement for fuzzy search is LIKE instead of '='. The wildcard character '%' could replace the missing letters. Add 'order by replace(,'','')' function at end can sort the result by the degree of proximity of the selected content, rather than by the original order of the primary. 'limit' statement is used to limit the length of the result, maximum show the first 20 result object.

Date: 6.7

12 Variation of background color

Features：The background of each page is either a gradient or a Mosaic of different colors.

Realization method：Use background_image and @keyframes to control the changing of the background

Date: 6.5

13 Use of SVG

Features：We implemented the ripple pattern at the bottom of the page using SVG class.

Realization method: The construction of ripples is achieved through the use of different attributes in the SVG class.

Date：6.5

14 Use of bookstrap

Features: Bookstrap makes it much easier to use CSS.

Realization method: We used Bookstrap to apply some CSS templates directly to speed up our progress.

Data: 6.8

Appendix

Demo

Before the real task, we made a demo project. We haven't learned ER diagrams or mapping, so there's no drawing or planning.
May 7th - May 15th

1. A login webpage that works – should be able to tell “wrong user name” and “wrong password”; if we login successfully, we enter the admin page;
2. An admin page that can view all the users’ basic information (so that we can try the page turning methods);
3. To do so, we need an admin table that works, a user table that works;
4. We also need two webpages in JSP, and login servlet modules for login and view functions;
5. Jdbc functions for checking the login person in the admin table, and extracting the information from user table.

Admin table (demo)

Admin number	Surname	Forename	password
18206096	Ma	Benteng	18206096

User table (demo)

ID	Surname	Forename	Screen name	gender
1100000001	Ma	Benteng	Ben	Male

Six persons, we will view them 3 person each page

DB backend interface

```
package com.example.jsp_demo;

import java.util.List;
import java.util.Map;

public interface DBBackendDemo {
    public boolean isAdmin(int adminID);
    public boolean adminLogin(int adminID, String password) throws PersonNotExistsException;
    public Map<String, String> getAdminInfo(int adminID) throws PersonNotExistsException;

    public boolean isUser(int userID);
    public boolean userLogin(int userID, String password) throws PersonNotExistsException;
    public Map<String, String> getUserInfo(int userID) throws PersonNotExistsException;
    public List<Map<String, String>> getUsersInfo(int[] userIDs) throws PersonNotExistsException;
}
```

Public Exception

```
package com.example.jsp_demo;

public class PersonNotExistsException extends RuntimeException{
    public PersonNotExistsException(String msg) {
        super(msg);
```

```
    }
}
```

==========

Table1 admins (add, delete, modify, remove user, reset password)

Use a primary key: admin number,

Other keys (not null): family name, first name. password.

Table2 users

Primary key: national id number.

Not null key: name, gender, *sexual orientation, state (single/in a relationship), headshot.

Other keys: email/WeChat/mobile location (province, city, district/county, village/streets), education level, occupation (type of work, detail), hobbies (maybe more than 1), average salary, love experience, slogan, introduction, expectation of the other half (importance ranks through the keys of other users; plus, a string for expression), *photos (no more than 5).

All the keys are related with visibility.

"like" keys, this user likes – up to 5 (it is better to be infinite); who like this user – up to 20? (need more consideration – should be infinite)

Table3 locations (city/province; district/county) (two tables)

serial numbers (primary) – place

maybe longitude and latitude?

Table4 – table10 allow add extra tuples:

Table4 universities/colleges/schools

serial numbers (primary) – name

Table5 jobs (type; specific) (two tables)

Serial numbers (primary) – name

Table6 hobbies

Serial numbers (primary) – name

(similar to the above – serial number as primary key + name)

Table7 food

Table8 place want to visit

Table9 music

Table10 films

Table11 logs

Time stamp + content

Functions

1. The new users can register in and full in his/her information.
    a) It refers to a specific web page
    b) It refers to updating user table
    c) It refers to fetching information from a few tables
    d) It refers to writing the log

2. The users can change their information.
    a) We can try to reuse functions in 1. as much as possible
    b) Writing to the log

3. *** THE MAIN FUNCTION** Give a list of matches for the user to recommend other people, including full visible information on a web page.
    a) A new webpage (or maybe give to the index page)
    b) Refers to several tables (centred at user table)
    c) * Uses match methods according to the rank this user is expecting
    d) User can see the information given by
    e) Take log

4. Users can choose people they like, one person can only “interest” in one person. Both sides can view this.
    a) An extra function added on the last page

b) User should be able to view the user he/she likes; and the users that like him/her

c) If two users both like each other, it should be shown specifically, and they may be allowed to view the email or wechat id of each other; others might not.

5. *Additionally, we do it if we have time – message system (I don't think we have the time)

6. The admins can view all the users' information. (add, delete, modify, remove user, reset password)

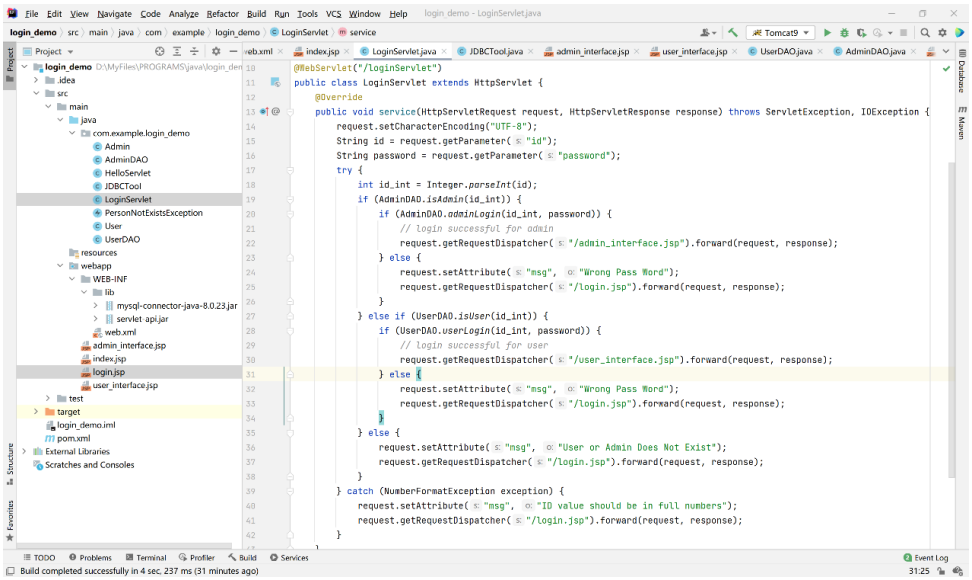
   a) We do it if we have time

   b) Log for any modification

5-13

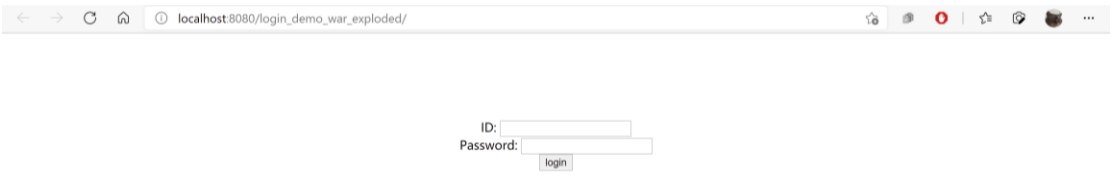
Demo have already became successful

By this week we've managed to login as a service user or an admin into the system, by entering user id and password onto a JSP web page. The login process is implemented using a servlet module and a database backend. They are now, all running.

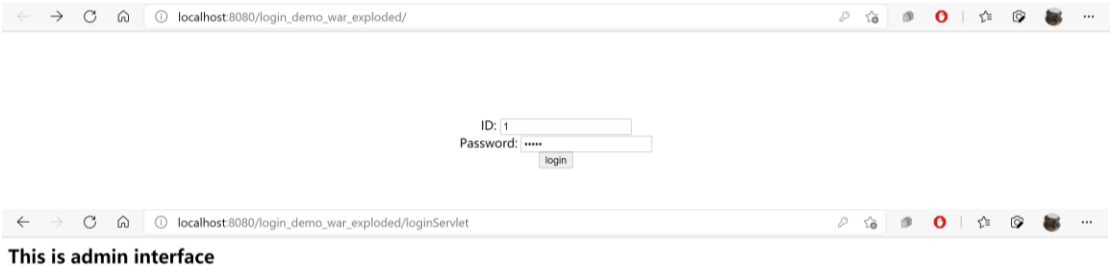


```
@WebServlet("/loginServlet")
public class LoginServlet extends HttpServlet {
    @Override
    public void service(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        request.setCharacterEncoding("UTF-8");
        String id = request.getParameter("id");
        String password = request.getParameter("password");
        try {
            int id_int = Integer.parseInt(id);
            if (AdminDAO.isAdmin(id_int)) {
                if (AdminDAO.adminLogin(id_int, password)) {
                    // login successful for admin
                    request.getRequestDispatcher("/admin_interface.jsp").forward(request, response);
                } else {
                    request.setAttribute("msg", "Wrong Pass Word");
                    request.getRequestDispatcher("/login.jsp").forward(request, response);
                }
            } else if (UserDAO.isUser(id_int)) {
                if (UserDAO.userLogin(id_int, password)) {
                    // login successful for user
                    request.getRequestDispatcher("/user_interface.jsp").forward(request, response);
                } else {
                    request.setAttribute("msg", "Wrong Pass Word");
                    request.getRequestDispatcher("/login.jsp").forward(request, response);
                }
            } else {
                request.setAttribute("msg", "User or Admin Does Not Exist");
                request.getRequestDispatcher("/login.jsp").forward(request, response);
            }
        } catch (NumberFormatException exception) {
            request.setAttribute("msg", "ID value should be in full numbers");
            request.getRequestDispatcher("/login.jsp").forward(request, response);
        }
```

Part of code for java servlet



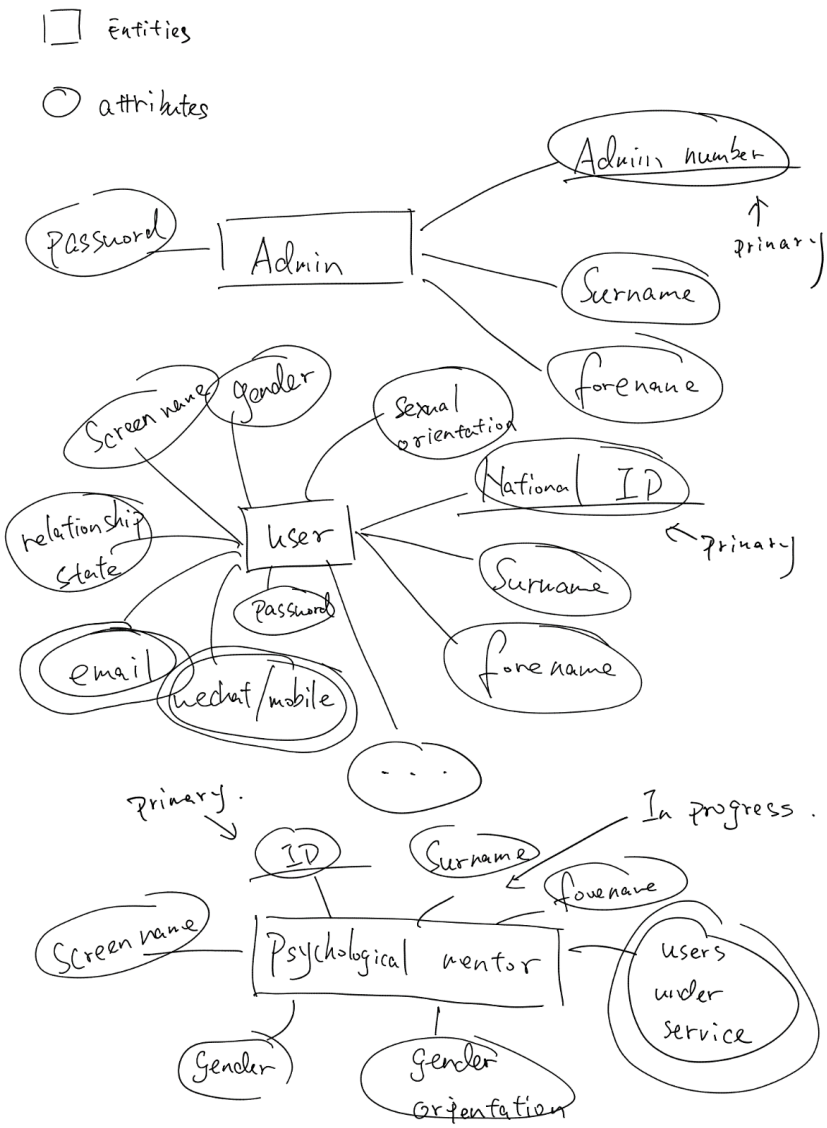
The rough resigned login page



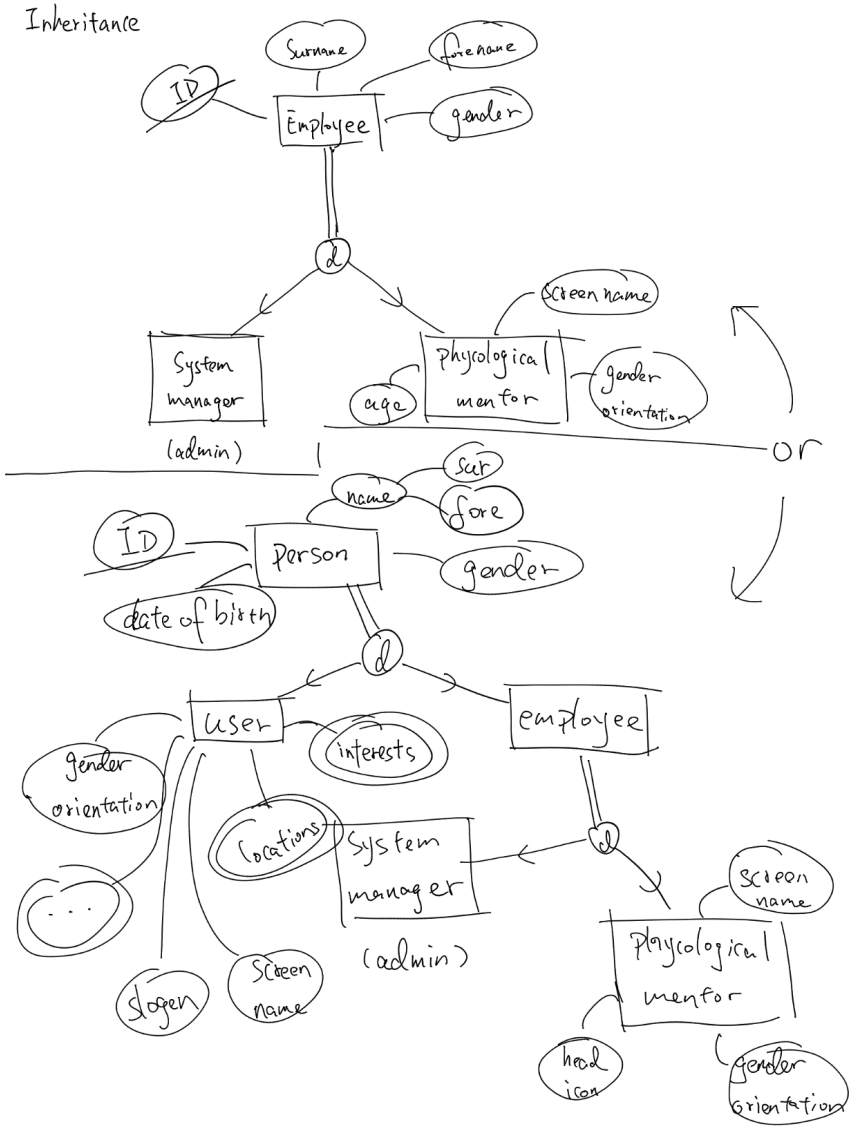
Entering the correct id and password will lead to admin or user's page

ER diagrams of our designed system

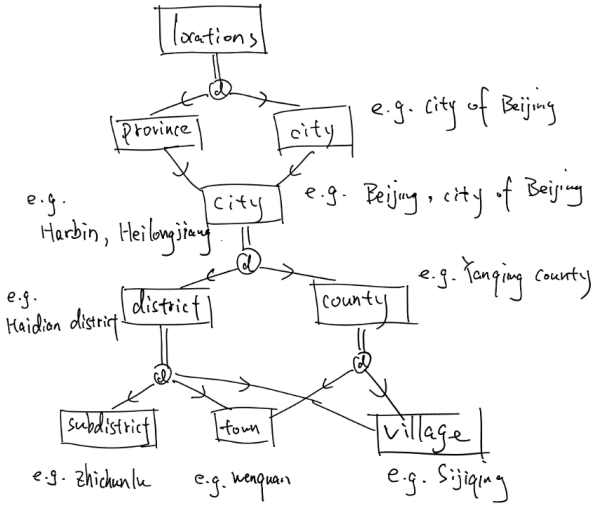
Designed Attributes for Admin, User and Mentor

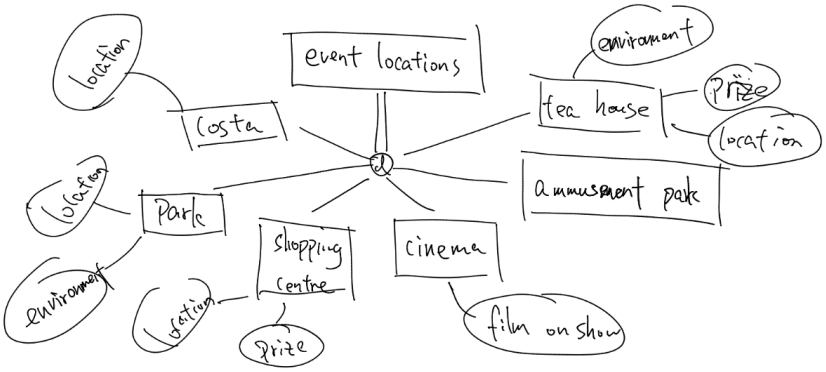


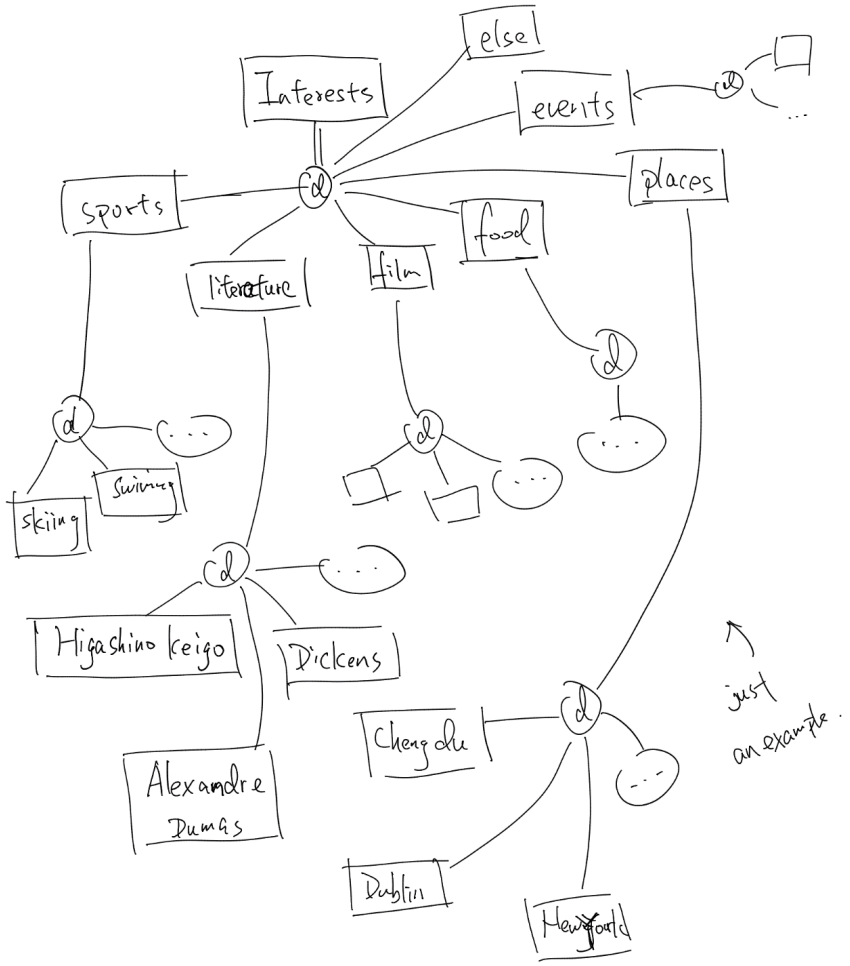
The hierarchical structure of the three types of people is

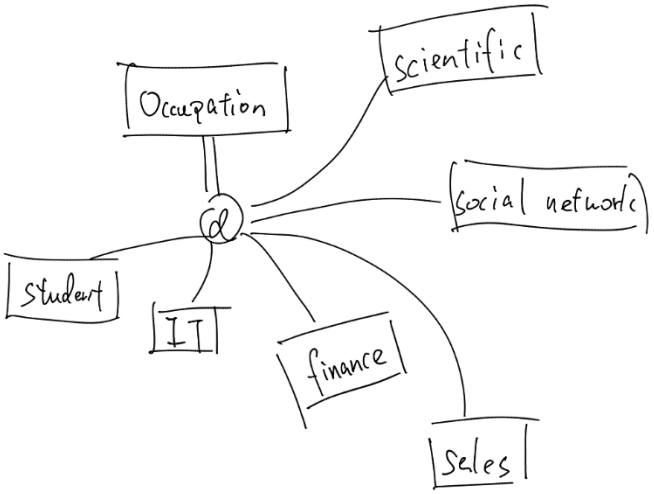


Other hierarchical structures

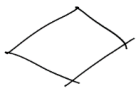

locations
d
province
city
e.g. city of Beijing
city
e.g. Beijing, city of Beijing
e.g.
Harbin, Heilongjiang
d
e.g. Yanqing county
e.g.
Haidian district
district
county
d
d
subdistrict
town
village
e.g. Zhichunlu
e.g. Wenquan
e.g. Sijiqing


location
event locations
environment
Costa
tea house
Prize
location
d
location
Park
amusement park
shopping
centre
cinema
environment
location
prize
film on show

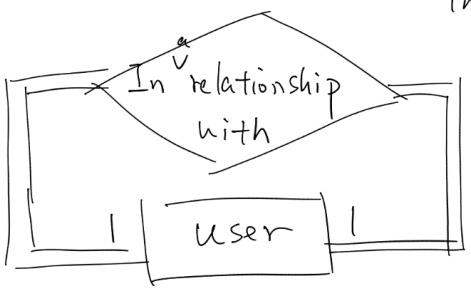




Relationships include

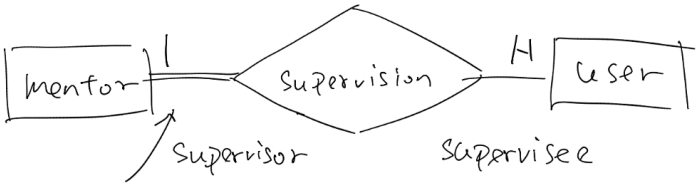


relationships.



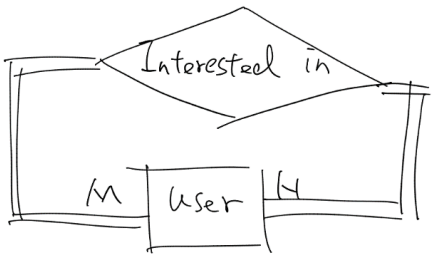
This must be 1:1 relationship.

optional

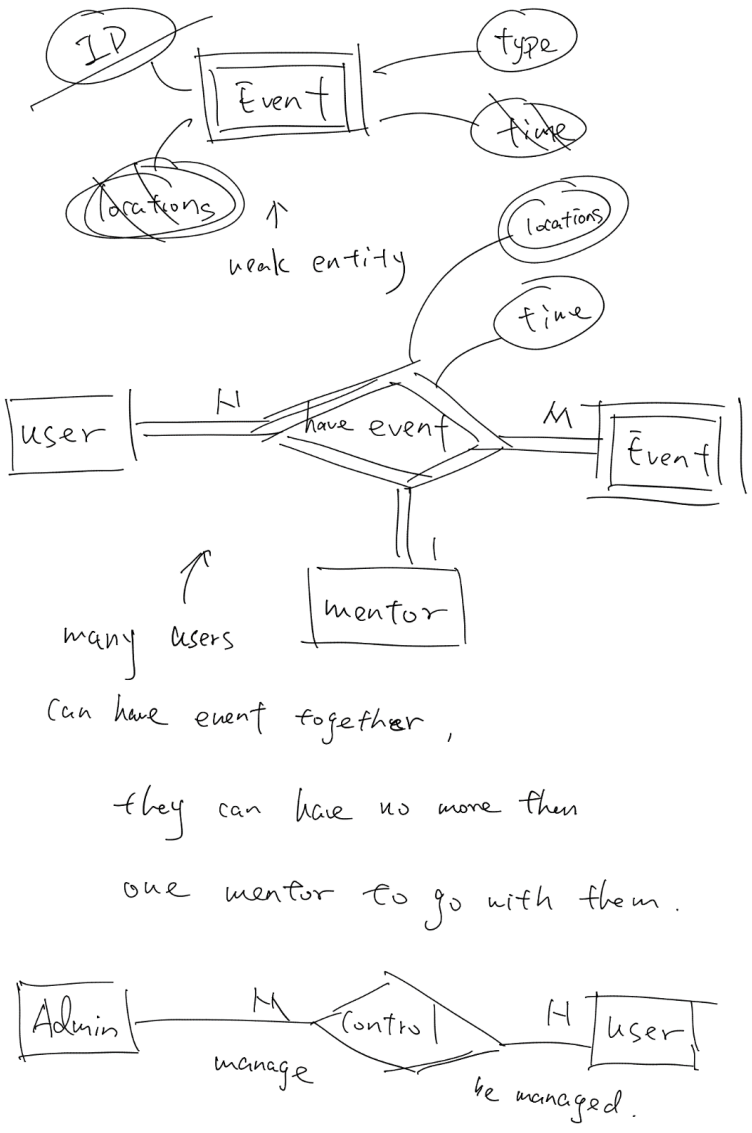


users don't have to (optional) supervised by a mentor

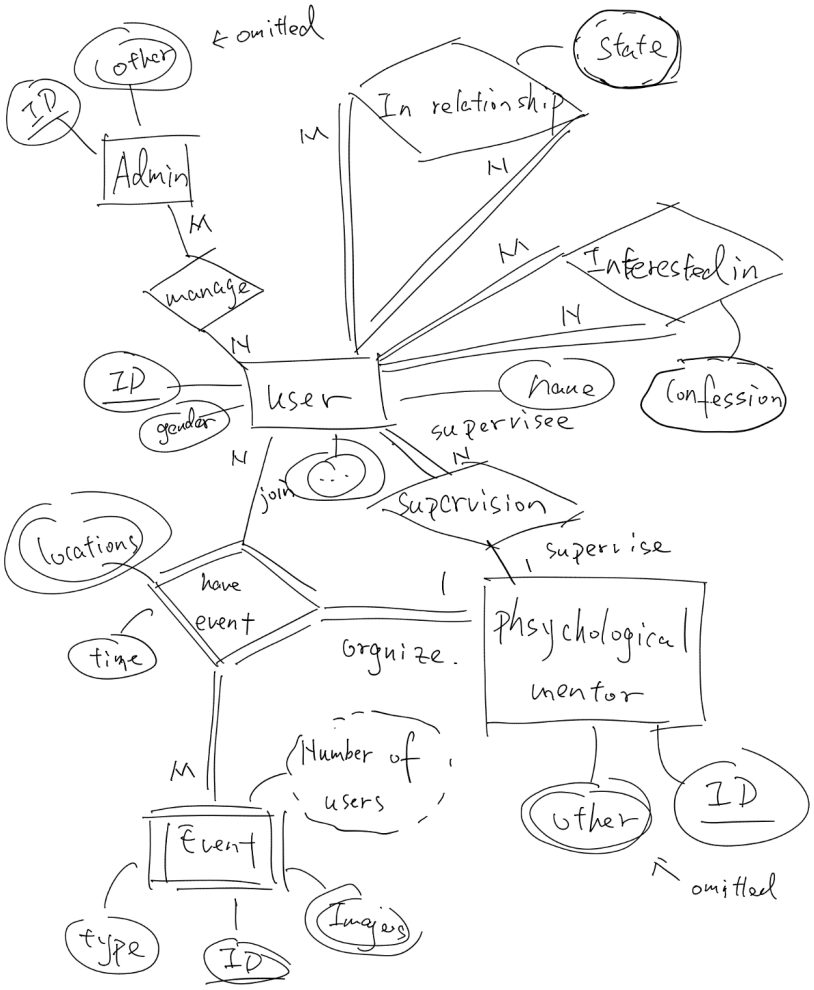
This is also a 1:M relationship



one user can be interested in many others



And the rough total relationship would be



Create Table in MySQL

the statement is shown below:

```
CREATE DATABASE groupwork;

use groupwork;

CREATE TABLE groupwork.person (
   SystemID int NOT NULL AUTO_INCREMENT,
   Surname varchar(64) DEFAULT NULL,
   Forename varchar(64) DEFAULT NULL,
   Gender enum ('male', 'female', 'secret') DEFAULT NULL,
   ScreenName char(64) DEFAULT NULL,
```

```
    HeadIcon varchar(128) DEFAULT NULL,
    password char(64) NOT NULL,
    PRIMARY KEY (SystemID)
)
ENGINE = INNODB,
CHARACTER SET utf8mb4,
COLLATE utf8mb4_0900_ai_ci;

ALTER TABLE groupwork.person
ADD UNIQUE INDEX ScreenName (ScreenName);

CREATE TABLE groupwork.labels (
    Serial int NOT NULL AUTO_INCREMENT,
    Locations varchar(64) DEFAULT NULL,
    Work varchar(64) DEFAULT NULL,
    Food varchar(64) DEFAULT NULL,
    Film varchar(64) DEFAULT NULL,
    Book varchar(64) DEFAULT NULL,
    Sport varchar(64) DEFAULT NULL,
    Activity varchar(64) DEFAULT NULL,
    PRIMARY KEY (Serial)
)
ENGINE = INNODB,
CHARACTER SET utf8mb4,
COLLATE utf8mb4_0900_ai_ci;

CREATE TABLE groupwork.employee (
    SystemID int NOT NULL,
    EmployeeID int NOT NULL AUTO_INCREMENT,
    PRIMARY KEY (EmployeeID)
)
ENGINE = INNODB,
CHARACTER SET utf8mb4,
COLLATE utf8mb4_0900_ai_ci;

ALTER TABLE groupwork.employee
ADD CONSTRAINT employee_ibfk_1 FOREIGN KEY (SystemID)
REFERENCES groupwork.person (SystemID) ON DELETE CASCADE ON UPDATE CASCADE;

CREATE TABLE groupwork.administrator (
    SystemID int NOT NULL,
    EmployeeID int NOT NULL,
    AdminNumber int NOT NULL AUTO_INCREMENT,
    PRIMARY KEY (AdminNumber)
```

```
)
ENGINE = INNODB,
AUTO_INCREMENT = 2,
CHARACTER SET utf8mb4,
COLLATE utf8mb4_0900_ai_ci;

ALTER TABLE groupwork.administrator
ADD CONSTRAINT administrator_ibfk_1 FOREIGN KEY (SystemID)
REFERENCES groupwork.person (SystemID) ON DELETE CASCADE ON UPDATE CASCADE;

ALTER TABLE groupwork.administrator
ADD CONSTRAINT administrator_ibfk_2 FOREIGN KEY (EmployeeID)
REFERENCES groupwork.employee (EmployeeID) ON DELETE CASCADE ON UPDATE
CASCADE;

CREATE TABLE groupwork.phsycological_mentor (
  SystemID int NOT NULL,
  EmployeeID int NOT NULL,
  MentorNumber int NOT NULL AUTO_INCREMENT,
  GenderOrientationInCharge enum ('homosexual', 'hetero') DEFAULT NULL,
  AgeRangeInRange int DEFAULT NULL,
  PRIMARY KEY (MentorNumber)
)
ENGINE = INNODB,
CHARACTER SET utf8mb4,
COLLATE utf8mb4_0900_ai_ci;

ALTER TABLE groupwork.phsycological_mentor
ADD CONSTRAINT phsycological_mentor_ibfk_1 FOREIGN KEY (SystemID)
REFERENCES groupwork.person (SystemID) ON DELETE CASCADE ON UPDATE CASCADE;

ALTER TABLE groupwork.phsycological_mentor
ADD CONSTRAINT phsycological_mentor_ibfk_2 FOREIGN KEY (EmployeeID)
REFERENCES groupwork.employee (EmployeeID) ON DELETE CASCADE ON UPDATE
CASCADE;

CREATE TABLE groupwork.user (
  SystemID int NOT NULL,
  UserID int NOT NULL AUTO_INCREMENT,
  Emailaddress varchar(128) DEFAULT NULL,
  Wechat varchar(64) DEFAULT NULL,
  GenderOrientation enum ('homosexual', 'hetero') DEFAULT NULL,
  DataOfBirth date DEFAULT NULL,
  Slogan varchar(512) DEFAULT NULL,
```

```
   Work int DEFAULT NULL,
   MentorID int DEFAULT NULL,
   PRIMARY KEY (UserID)
)
ENGINE = INNODB,
CHARACTER SET utf8mb4,
COLLATE utf8mb4_0900_ai_ci;

ALTER TABLE groupwork.user
ADD UNIQUE INDEX Emailaddress (Emailaddress);

ALTER TABLE groupwork.user
ADD UNIQUE INDEX Wechat (Wechat);

ALTER TABLE groupwork.user
ADD CONSTRAINT user_ibfk_1 FOREIGN KEY (SystemID)
REFERENCES groupwork.person (SystemID) ON DELETE CASCADE ON UPDATE CASCADE;

ALTER TABLE groupwork.user
ADD CONSTRAINT user_ibfk_2 FOREIGN KEY (MentorID)
REFERENCES groupwork.phsycological_mentor (MentorNumber) ON DELETE CASCADE
ON UPDATE CASCADE;

ALTER TABLE groupwork.user
ADD CONSTRAINT user_ibfk_3 FOREIGN KEY (Work)
REFERENCES groupwork.labels (Serial) ON DELETE CASCADE ON UPDATE CASCADE;

CREATE TABLE groupwork.date (
   Uid1 int NOT NULL,
   Uid2 int NOT NULL,
   HHDDMMYY datetime NOT NULL DEFAULT '0000-00-00 00:00:00',
   GeographicalLocation varchar(256) DEFAULT NULL,
   PRIMARY KEY (Uid1, Uid2, HHDDMMYY)
)
ENGINE = INNODB,
CHARACTER SET utf8mb4,
COLLATE utf8mb4_0900_ai_ci;

ALTER TABLE groupwork.date
ADD CONSTRAINT date_ibfk_1 FOREIGN KEY (Uid1)
REFERENCES groupwork.user (UserID) ON DELETE CASCADE ON UPDATE CASCADE;

ALTER TABLE groupwork.date
ADD CONSTRAINT date_ibfk_2 FOREIGN KEY (Uid2)
```

```
REFERENCES groupwork.user (UserID) ON DELETE CASCADE ON UPDATE CASCADE;

CREATE TABLE groupwork.event_location (
   LocationID int NOT NULL AUTO_INCREMENT,
   LocationType enum ('coffee/tea house', 'bar', 'restaurant', 'shopping centre', 'park', 'cinema',
'theatre') DEFAULT NULL,
   GeographicalLocation varchar(256) DEFAULT NULL,
   ManagerID int DEFAULT NULL,
   StartTime date DEFAULT NULL,
   PRIMARY KEY (LocationID)
)
ENGINE = INNODB,
CHARACTER SET utf8mb4,
COLLATE utf8mb4_0900_ai_ci;

ALTER TABLE groupwork.event_location
ADD CONSTRAINT event_location_ibfk_1 FOREIGN KEY (ManagerID)
REFERENCES groupwork.phsycological_mentor (MentorNumber) ON DELETE CASCADE
ON UPDATE CASCADE;

CREATE TABLE groupwork.event (
   LocationID int NOT NULL AUTO_INCREMENT,
   Time datetime NOT NULL,
   Number_of_participants int DEFAULT NULL,
   Activities char(15) DEFAULT NULL,
   PRIMARY KEY (LocationID, Time)
)
ENGINE = INNODB,
CHARACTER SET utf8mb4,
COLLATE utf8mb4_0900_ai_ci;

ALTER TABLE groupwork.event
ADD CONSTRAINT event_ibfk_1 FOREIGN KEY (LocationID)
REFERENCES groupwork.event_location (LocationID) ON DELETE CASCADE ON UPDATE
CASCADE;

CREATE TABLE `join_event` (
   `Uid` int NOT NULL,
   `EventLocationID` int NOT NULL,
   `Time` datetime NOT NULL,
   PRIMARY KEY (`EventLocationID`,`Uid`,`Time`) USING BTREE,
   KEY `Uid` (`Uid`),
   KEY `Time` (`Time`),
   KEY `EventLocationID` (`EventLocationID`,`Time`),
```

```
  CONSTRAINT `join_event_ibfk_1` FOREIGN KEY (`Uid`) REFERENCES `user` (`UserID`) ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT `join_event_ibfk_2` FOREIGN KEY (`EventLocationID`) REFERENCES `event_location` (`LocationID`) ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT `join_event_ibfk_3` FOREIGN KEY (`EventLocationID`, `Time`) REFERENCES `event` (`LocationID`, `Time`) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;

CREATE TABLE groupwork.likes (
  Uid1 int NOT NULL,
  Uid2 int NOT NULL,
  Confession varchar(512) DEFAULT NULL,
  PRIMARY KEY (Uid1, Uid2)
)
ENGINE = INNODB,
CHARACTER SET utf8mb4,
COLLATE utf8mb4_0900_ai_ci;

ALTER TABLE groupwork.likes
ADD CONSTRAINT likes_ibfk_1 FOREIGN KEY (Uid1)
REFERENCES groupwork.user (UserID) ON DELETE CASCADE ON UPDATE CASCADE;

ALTER TABLE groupwork.likes
ADD CONSTRAINT likes_ibfk_2 FOREIGN KEY (Uid2)
REFERENCES groupwork.user (UserID) ON DELETE CASCADE ON UPDATE CASCADE;

CREATE TABLE groupwork.relationship (
  Uid1 int NOT NULL,
  Uid2 int NOT NULL,
  RelationState enum ('acquaintance', 'ambiguous', 'boyfriend/girlfriend', 'engaged', 'married') DEFAULT NULL,
  PRIMARY KEY (Uid1, Uid2)
)
ENGINE = INNODB,
CHARACTER SET utf8mb4,
COLLATE utf8mb4_0900_ai_ci;

ALTER TABLE groupwork.relationship
ADD CONSTRAINT relationship_ibfk_1 FOREIGN KEY (Uid1)
REFERENCES groupwork.user (UserID) ON DELETE CASCADE ON UPDATE CASCADE;

ALTER TABLE groupwork.relationship
ADD CONSTRAINT relationship_ibfk_2 FOREIGN KEY (Uid2)
```

```
REFERENCES groupwork.user (UserID) ON DELETE CASCADE ON UPDATE CASCADE;

CREATE TABLE groupwork.user_photos (
   Uid int NOT NULL AUTO_INCREMENT,
   `PhotoPath(in the system)` varchar(128) DEFAULT NULL,
   PRIMARY KEY (Uid)
)
ENGINE = INNODB,
CHARACTER SET utf8mb4,
COLLATE utf8mb4_0900_ai_ci;

ALTER TABLE groupwork.user_photos
ADD CONSTRAINT user_photos_ibfk_1 FOREIGN KEY (Uid)
REFERENCES groupwork.user (UserID) ON DELETE CASCADE ON UPDATE CASCADE;

CREATE TABLE groupwork.books (
   Bid int NOT NULL,
   Uid int NOT NULL,
   PRIMARY KEY (Bid, Uid)
)
ENGINE = INNODB,
CHARACTER SET utf8mb4,
COLLATE utf8mb4_0900_ai_ci;

ALTER TABLE groupwork.books
ADD CONSTRAINT books_ibfk_1 FOREIGN KEY (Bid)
REFERENCES groupwork.labels (Serial) ON DELETE CASCADE ON UPDATE CASCADE;

ALTER TABLE groupwork.books
ADD CONSTRAINT books_ibfk_2 FOREIGN KEY (Uid)
REFERENCES groupwork.user (UserID) ON DELETE CASCADE ON UPDATE CASCADE;

CREATE TABLE groupwork.films (
   Fid int NOT NULL,
   Uid int NOT NULL,
   PRIMARY KEY (Fid, Uid)
)
ENGINE = INNODB,
CHARACTER SET utf8mb4,
COLLATE utf8mb4_0900_ai_ci;

ALTER TABLE groupwork.films
ADD CONSTRAINT films_ibfk_1 FOREIGN KEY (Fid)
REFERENCES groupwork.labels (Serial) ON DELETE CASCADE ON UPDATE CASCADE;
```

```
ALTER TABLE groupwork.films
ADD CONSTRAINT films_ibfk_2 FOREIGN KEY (Uid)
REFERENCES groupwork.user (UserID) ON DELETE CASCADE ON UPDATE CASCADE;

CREATE TABLE groupwork.food (
   Fid int NOT NULL,
   Uid int NOT NULL,
   PRIMARY KEY (Fid, Uid)
)
ENGINE = INNODB,
CHARACTER SET utf8mb4,
COLLATE utf8mb4_0900_ai_ci;

ALTER TABLE groupwork.food
ADD CONSTRAINT food_ibfk_1 FOREIGN KEY (Fid)
REFERENCES groupwork.labels (Serial) ON DELETE CASCADE ON UPDATE CASCADE;

ALTER TABLE groupwork.food
ADD CONSTRAINT food_ibfk_2 FOREIGN KEY (Uid)
REFERENCES groupwork.user (UserID) ON DELETE CASCADE ON UPDATE CASCADE;

CREATE TABLE groupwork.location (
   Lid int NOT NULL,
   Uid int NOT NULL,
   PRIMARY KEY (Lid, Uid)
)
ENGINE = INNODB,
CHARACTER SET utf8mb4,
COLLATE utf8mb4_0900_ai_ci;

ALTER TABLE groupwork.location
ADD CONSTRAINT location_ibfk_1 FOREIGN KEY (Lid)
REFERENCES groupwork.labels (Serial) ON DELETE CASCADE ON UPDATE CASCADE;

ALTER TABLE groupwork.location
ADD CONSTRAINT location_ibfk_2 FOREIGN KEY (Uid)
REFERENCES groupwork.user (UserID) ON DELETE CASCADE ON UPDATE CASCADE;

CREATE TABLE groupwork.sports (
   Sid int NOT NULL,
   Uid int NOT NULL,
   PRIMARY KEY (Sid, Uid)
)
```

ENGINE = INNODB,
CHARACTER SET utf8mb4,
COLLATE utf8mb4_0900_ai_ci;

ALTER TABLE groupwork.sports
ADD CONSTRAINT sports_ibfk_1 FOREIGN KEY (Sid)
REFERENCES groupwork.labels (Serial) ON DELETE CASCADE ON UPDATE CASCADE;

ALTER TABLE groupwork.sports
ADD CONSTRAINT sports_ibfk_2 FOREIGN KEY (Uid)
REFERENCES groupwork.user (UserID) ON DELETE CASCADE ON UPDATE CASCADE;

USE groupwork;

CREATE VIEW UserPerson AS SELECT person.SystemID, person.Surname, person.Forename, person.Gender, person.ScreenName,
person.HeadIcon, person.password, user.UserID, user.Emailaddress, user.Wechat, user.GenderOrientation, user.DataOfBirth,
user.Slogan, user.Work, user.MentorID FROM user inner join person on user.SystemID = person.SystemID;

Final Tables.

```
mysql> DESCRIBE administrator;
+-------------+------+------+-----+---------+----------------+
| Field       | Type | Null | Key | Default | Extra          |
+-------------+------+------+-----+---------+----------------+
| SystemID    | int  | NO   | MUL | NULL    |                |
| EmployeeID  | int  | NO   | MUL | NULL    |                |
| AdminNumber | int  | NO   | PRI | NULL    | auto_increment |
+-------------+------+------+-----+---------+----------------+
3 rows in set (0.01 sec)
```

```
mysql> DESCRIBE books;
+-------+------+------+-----+---------+-------+
| Field | Type | Null | Key | Default | Extra |
+-------+------+------+-----+---------+-------+
| Bid   | int  | NO   | PRI | NULL    |       |
| Uid   | int  | NO   | PRI | NULL    |       |
+-------+------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

```
mysql> DESCRIBE date;
+---------------------+-------------+------+-----+---------------------+-------+
| Field               | Type        | Null | Key | Default             | Extra |
+---------------------+-------------+------+-----+---------------------+-------+
| Uid1                | int         | NO   | PRI | NULL                |       |
| Uid2                | int         | NO   | PRI | NULL                |       |
| HHDDMMYY            | datetime    | NO   | PRI | 0000-00-00 00:00:00 |       |
| GeographicalLocation | varchar(30) | YES  |     | NULL                |       |
+---------------------+-------------+------+-----+---------------------+-------+
4 rows in set (0.00 sec)
```

```
mysql> DESCRIBE employee;
+-------------+------+------+-----+---------+----------------+
| Field       | Type | Null | Key | Default | Extra          |
+-------------+------+------+-----+---------+----------------+
| SystemID    | int  | NO   | MUL | NULL    |                |
| EmployeeID  | int  | NO   | PRI | NULL    | auto_increment |
+-------------+------+------+-----+---------+----------------+
2 rows in set (0.00 sec)
```

```
mysql> DESCRIBE event;
+------------------------+----------+------+-----+---------+----------------+
| Field                  | Type     | Null | Key | Default | Extra          |
+------------------------+----------+------+-----+---------+----------------+
| LocationID             | int      | NO   | PRI | NULL    | auto_increment |
| Time                   | datetime | NO   | PRI | NULL    |                |
| Number of participants | int      | YES  |     | NULL    |                |
| Activities             | char(15) | YES  |     | NULL    |                |
+------------------------+----------+------+-----+---------+----------------+
4 rows in set (0.00 sec)
```

```
mysql> DESCRIBE event_location;
+--------------------+-----------------------------------------------------------------------------------------+------+-----+---------+----------------+
| Field              | Type                                                                                    | Null | Key | Default | Extra          |
+--------------------+-----------------------------------------------------------------------------------------+------+-----+---------+----------------+
| LocationID         | int                                                                                     | NO   | PRI | NULL    | auto_increment |
| LocationType       | enum('coffee/tea house','bar','restaurant','shopping centre','park','cinema','theatre') | YES  |     | NULL    |                |
| GeographicalLocation | varchar(30)                                                                           | YES  |     | NULL    |                |
| ManagerID          | int                                                                                     | YES  | MUL | NULL    |                |
| StartTime          | date                                                                                    | YES  |     | NULL    |                |
+--------------------+-----------------------------------------------------------------------------------------+------+-----+---------+----------------+
5 rows in set (0.00 sec)
```

```
mysql> DESCRIBE films;
+--------+------+------+-----+---------+-------+
| Field  | Type | Null | Key | Default | Extra |
+--------+------+------+-----+---------+-------+
| Fid    | int  | NO   | PRI | NULL    |       |
| Uid    | int  | NO   | PRI | NULL    |       |
+--------+------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

```
mysql> DESCRIBE food;
+--------+------+------+-----+---------+-------+
| Field  | Type | Null | Key | Default | Extra |
+--------+------+------+-----+---------+-------+
| Fid    | int  | NO   | PRI | NULL    |       |
| Uid    | int  | NO   | PRI | NULL    |       |
+--------+------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

```
mysql> describe join_event;
+-----------------+----------+------+-----+---------+-------+
| Field           | Type     | Null | Key | Default | Extra |
+-----------------+----------+------+-----+---------+-------+
| Uid             | int      | NO   | PRI | NULL    |       |
| EventLocationID | int      | NO   | PRI | NULL    |       |
| Time            | datetime | NO   | PRI | NULL    |       |
+-----------------+----------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

```
mysql> DESCRIBE labels;
+-----------+-------------+------+-----+---------+----------------+
| Field     | Type        | Null | Key | Default | Extra          |
+-----------+-------------+------+-----+---------+----------------+
| Serial    | int         | NO   | PRI | NULL    | auto_increment |
| Locations | varchar(10) | YES  |     | NULL    |                |
| Work      | varchar(10) | YES  |     | NULL    |                |
| Food      | varchar(10) | YES  |     | NULL    |                |
| Film      | varchar(10) | YES  |     | NULL    |                |
| Book      | varchar(10) | YES  |     | NULL    |                |
| Sport     | varchar(10) | YES  |     | NULL    |                |
| Activity  | varchar(10) | YES  |     | NULL    |                |
+-----------+-------------+------+-----+---------+----------------+
8 rows in set (0.00 sec)
```

```
mysql> DESCRIBE likes;
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| Uid1       | int         | NO   | PRI | NULL    |       |
| Uid2       | int         | NO   | PRI | NULL    |       |
| Confession | varchar(40) | YES  |     | NULL    |       |
+------------+-------------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

```
mysql> DESCRIBE location;
+--------+------+------+-----+---------+-------+
| Field  | Type | Null | Key | Default | Extra |
+--------+------+------+-----+---------+-------+
| Lid    | int  | NO   | PRI | NULL    |       |
| Uid    | int  | NO   | PRI | NULL    |       |
+--------+------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

```
mysql> DESCRIBE person;
+------------+---------------------------------+------+-----+---------+----------------+
| Field      | Type                            | Null | Key | Default | Extra          |
+------------+---------------------------------+------+-----+---------+----------------+
| SystemID   | int                             | NO   | PRI | NULL    | auto_increment |
| Surname    | varchar(10)                     | YES  |     | NULL    |                |
| Forename   | varchar(10)                     | YES  |     | NULL    |                |
| Gender     | enum('male','female','secret')  | YES  |     | NULL    |                |
| ScreenName | varchar(20)                     | YES  | UNI | NULL    |                |
| HeadIcon   | varchar(30)                     | YES  |     | NULL    |                |
| password   | char(30)                        | NO   |     | NULL    |                |
+------------+---------------------------------+------+-----+---------+----------------+
7 rows in set (0.00 sec)
```

The report for every week:

4-25

Table1 admins (add, delete, modify, remove user, reset password)
Use a primary key: admin number,
Other keys (not null): family name, first name. password.

Table2 users
Primary key: national id number.
Not null key: gender, *sexual orientation, state (single/in a relationship), headshot.
Other keys: email/WeChat/mobile location (province, city, district/county, village/streets),

education level, occupation (type of work, detail), hobbies (maybe more than 1), average salary, love experience, slogan, introduction, expectation of the other half, *photos.
All the keys are related with visibility.

Table3 locations

Table4 universities/colleges/schools

Table5 jobs

Table6 hobbies

Table7 relatives?

Table8 expectations (match result?)

Table9 logs

Table10 images

Table11 interest table (interested on someone else)

Functions

The new users can register in and full in his/her information.

The users can change their information.

The admins can view all the users' information. (add, delete, modify, remove user, reset password)

Give a list of matches for the user to recommend other people, including full visible information on a web page.

Users can choose people they like, one person can only "interest" in one person. Both sides can view this.

5-7

Task this week - before May 15th

6. A login webpage that works – should be able to tell "wrong user name" and

"wrong password"; if we login successfully, we enter the admin page;

7. An admin page that can view all the users' basic information (so that we can try the page turning methods);
8. To do so, we need an admin table that works, a user table that works;
9. We also need two webpages in JSP, and login servlet modules for login and view functions;
10. Jdbc functions for checking the login person in the admin table, and extracting the information from user table.

Admin table (demo)

Admin number	Surname	Forename	password
18206096	Ma	Benteng	18206096

User table (demo)

ID	Surname	Forename	Screen name	gender
1100000001	Ma	Benteng	Ben	Male

Six persons, we will view them 3 person each page

DB backend interface

```
package com.example.jsp_demo;

import java.util.List;
import java.util.Map;

public interface DBBackendDemo {
    public boolean isAdmin(int adminID);
    public boolean adminLogin(int adminID, String password) throws PersonNotExistsException;
    public Map<String, String> getAdminInfo(int adminID) throws PersonNotExistsException;

    public boolean isUser(int userID);
    public boolean userLogin(int userID, String password) throws PersonNotExistsException;
    public Map<String, String> getUserInfo(int userID) throws PersonNotExistsException;
    public List<Map<String, String>> getUsersInfo(int[] userIDs) throws PersonNotExistsException;
}
```

Public Exception

```
package com.example.jsp_demo;

public class PersonNotExistsException extends RuntimeException{
    public PersonNotExistsException(String msg) {
        super(msg);
    }
}
```

==========

Table1 admins (add, delete, modify, remove user, reset password)

Use a primary key: admin number,

Other keys (not null): family name, first name. password.

Table2 users

Primary key: national id number.

Not null key: name, gender, *sexual orientation, state (single/in a relationship), headshot.

Other keys: email/WeChat/mobile location (province, city, district/county, village/streets), education level, occupation (type of work, detail), hobbies (maybe more than 1), average salary, love experience, slogan, introduction, expectation of the other half (importance ranks through the keys of other users; plus, a string for expression), *photos (no more than 5).

All the keys are related with visibility.

"like" keys, this user likes – up to 5 (it is better to be infinite); who like this user – up to 20? (need more consideration – should be infinite)

Table3 locations (city/province; district/county) (two tables)

serial numbers (primary) – place

maybe longitude and latitude?

Table4 – table10 allow add extra tuples:

Table4 universities/colleges/schools

serial numbers (primary) – name

Table5 jobs (type; specific) (two tables)

Serial numbers (primary) – name

Table6 hobbies

Serial numbers (primary) – name

(similar to the above – serial number as primary key + name)

Table7 food

Table8 place want to visit

Table9 music

Table10 films

Table11 logs

Time stamp + content

Functions

7. The new users can register in and full in his/her information.
    a) It refers to a specific web page
    b) It refers to updating user table
    c) It refers to fetching information from a few tables
    d) It refers to writing the log

8. The users can change their information.
    a) We can try to reuse functions in 1. as much as possible
    b) Writing to the log

9. *** THE MAIN FUNCTION** Give a list of matches for the user to recommend other people, including full visible information on a web page.
    a) A new webpage (or maybe give to the index page)
    b) Refers to several tables (centred at user table)
    c) *** Uses match methods according to the rank this user is expecting**
    d) User can see the information given by
    e) Take log

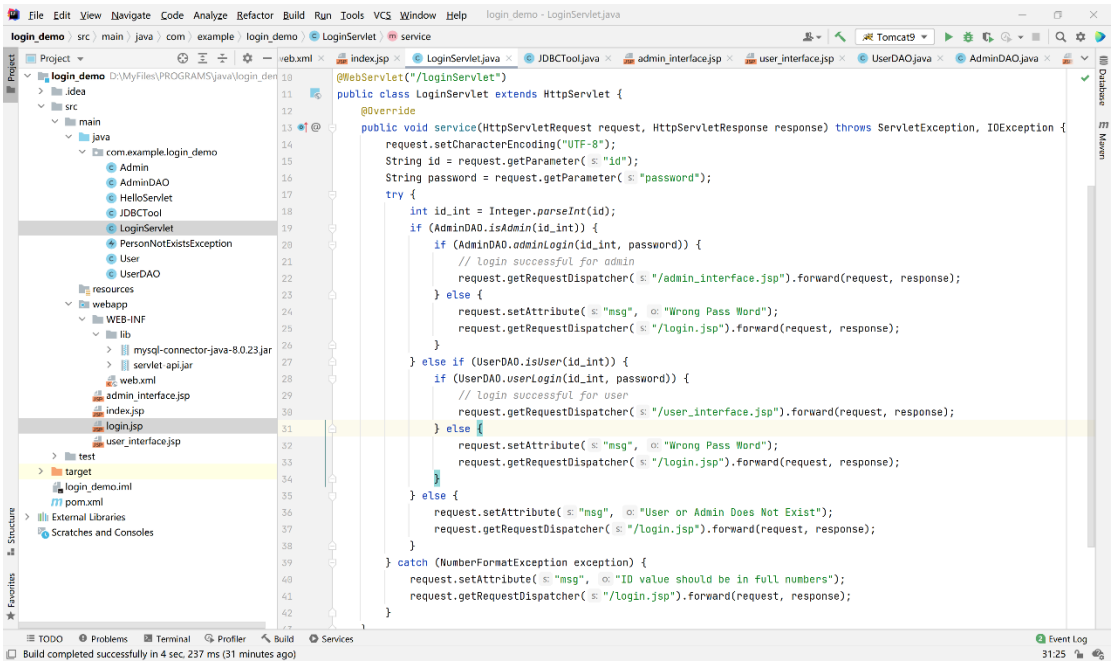
10. Users can choose people they like, one person can only "interest" in one person. Both sides can view this.
    a) An extra function added on the last page
    b) User should be able to view the user he/she likes; and the users that like him/her
    c) If two users both like each other, it should be shown specifically, and they may be allowed to view the email or wechat id of each other; others might not.

11. *Additionally, we do it if we have time – message system (I don't think we have the time)

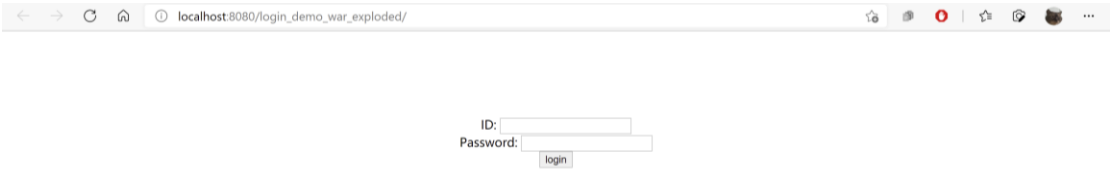
12. The admins can view all the users' information. (add, delete, modify, remove user, reset password)
    a) We do it if we have time
    b) Log for any modification

5-16

By this week we've managed to login as a service user or an admin into the system, by entering user id and password onto a JSP web page. The login process is implemented using a servlet module and a database backend. They are now, all running.



Part of code for java servlet



The rough resigned login page

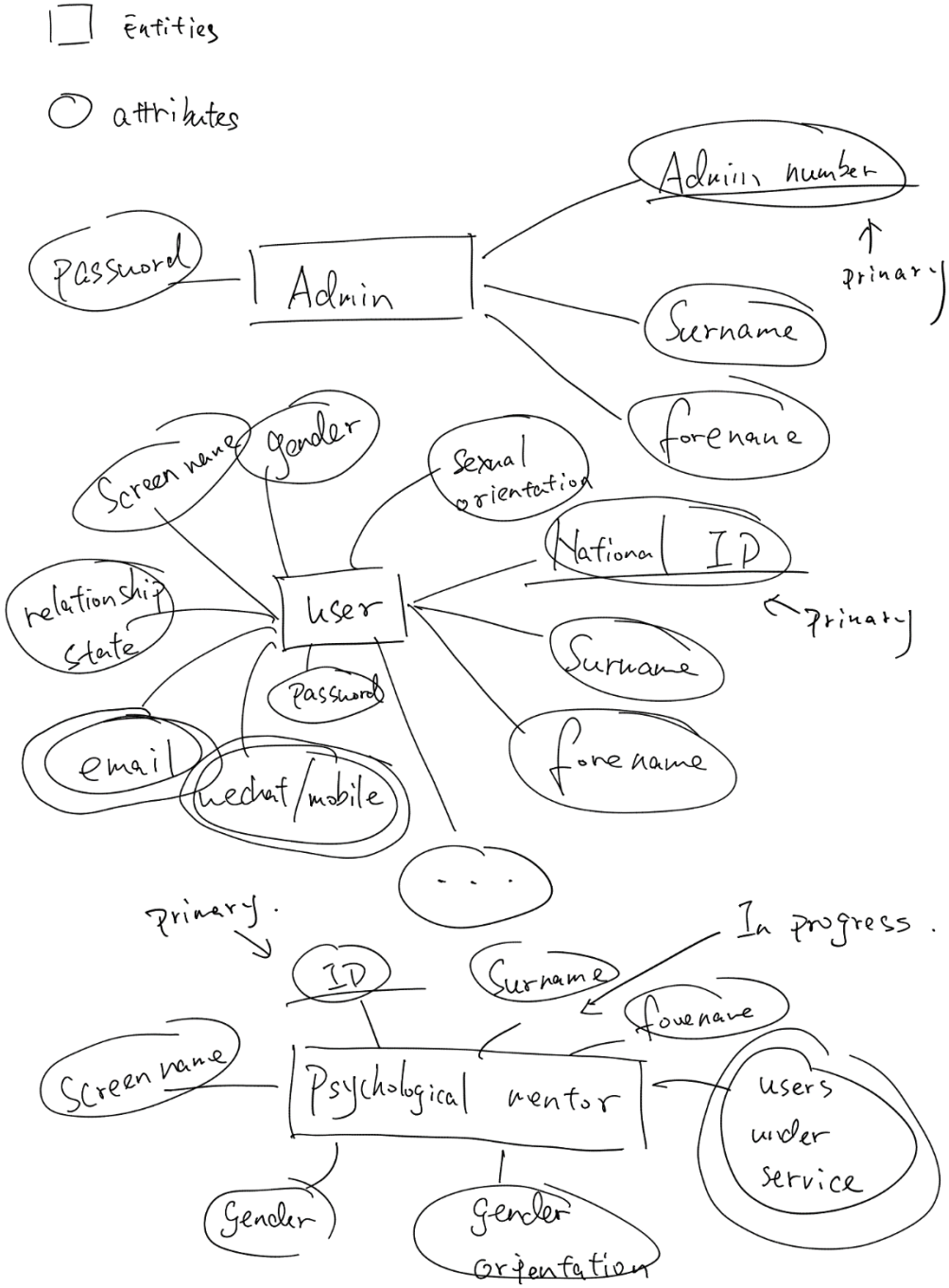
Entering the correct id and password will lead to admin or user's page

Plan for next week

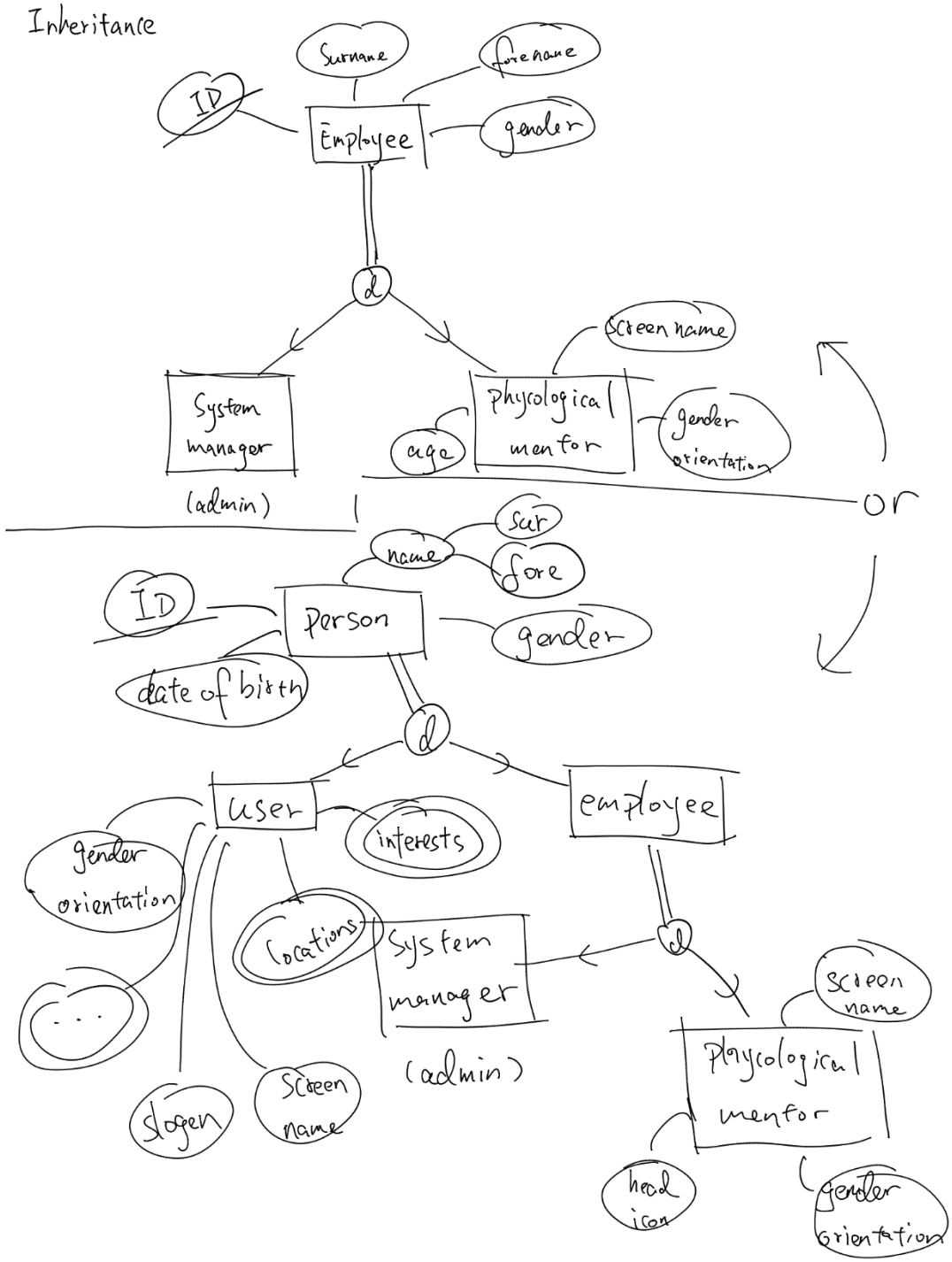
1. Decorate this login page with CSS
2. Finish these demo pages for user and admin
3. Start to implement the formal version of out data system according to or design (shown in ER diagrams)

ER diagrams of our designed system

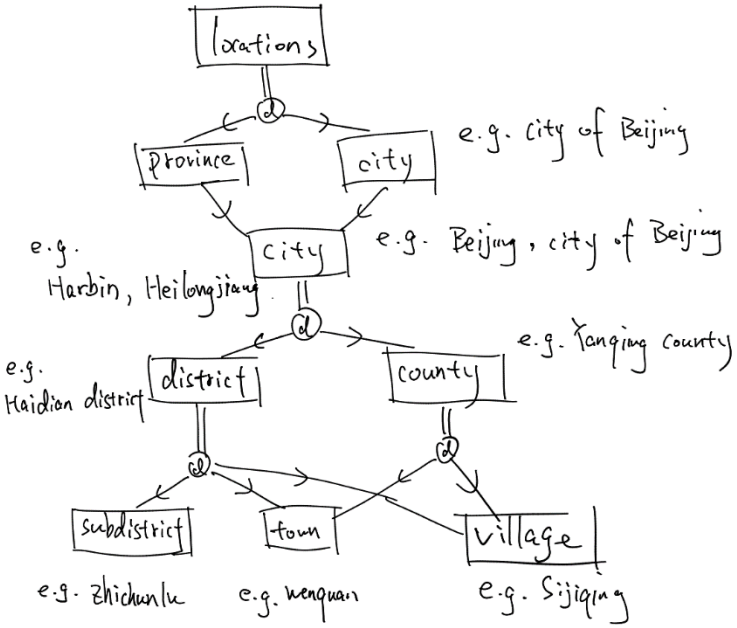
Designed Attributes for Admin, User and Mentor

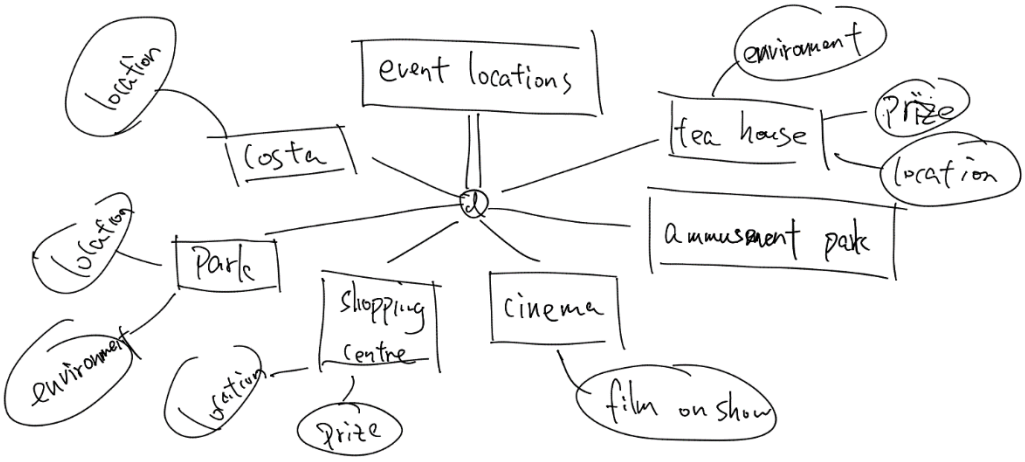


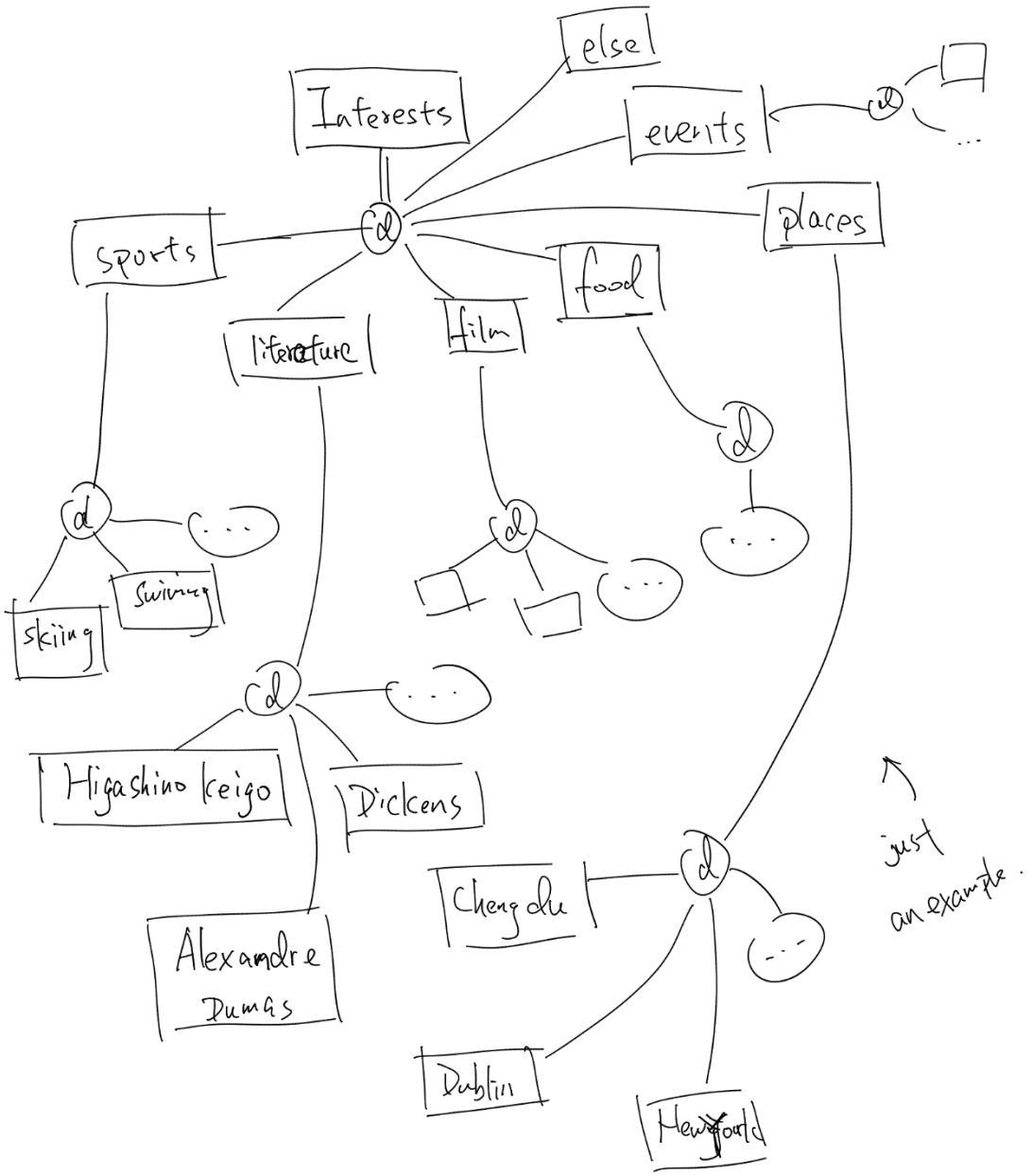
The hierarchical structure of the three types of people is

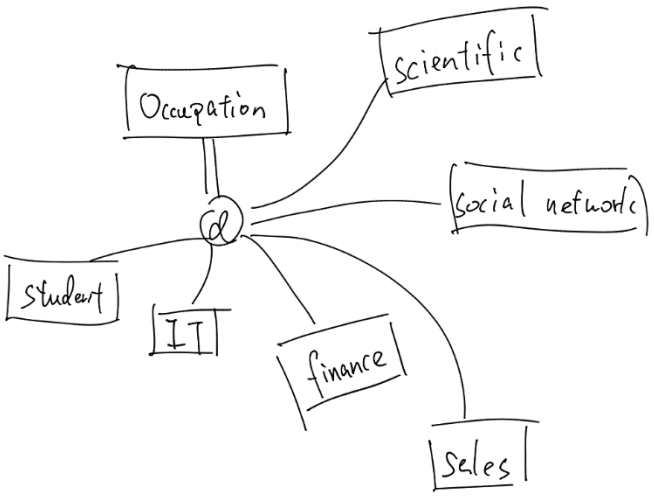


Other hierarchical structures

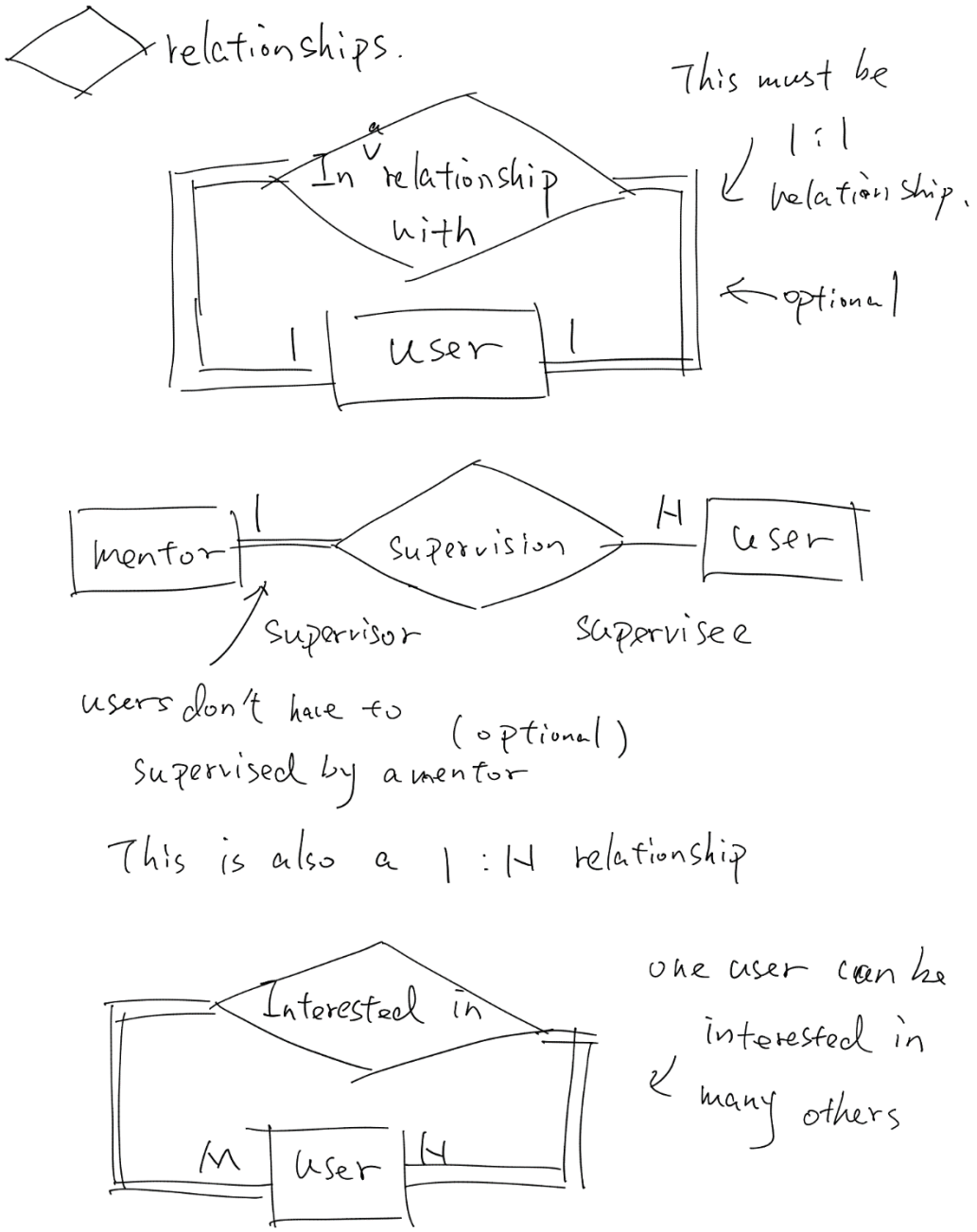

locations
province
city
e.g. city of Beijing
city
e.g. Beijing, city of Beijing
e.g.
Harbin, Heilongjiang
e.g. Yanqing County
e.g.
Haidian district
district
county
subdistrict
town
village
e.g. Zhichunlu
e.g. Wenquan
e.g. Sijiqing

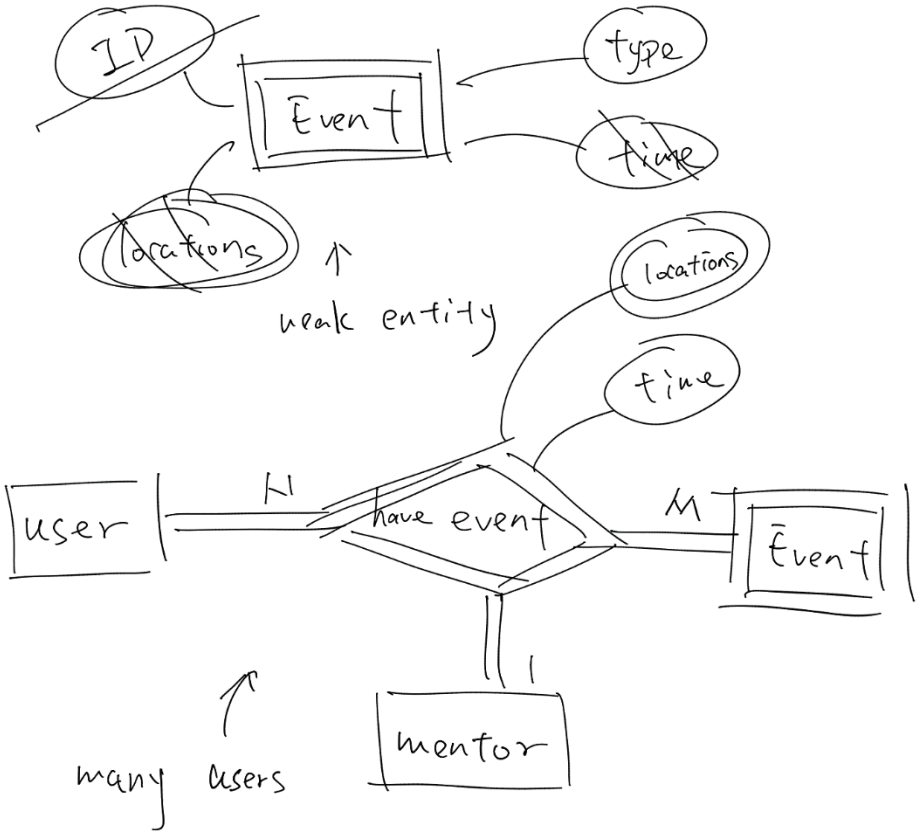

location
event locations
environment
costa
tea house
prize
location
location
park
amusement park
shopping centre
cinema
environment
location
prize
film on show


Interests
else
events
places
sports
food
literature
film
skiing
Swimming
Higashino Keigo
Dickens
Alexandre Dumas
Chengdu
Dublin
New York
just an example.


Occupation
scientific
social network
student
IT
finance
sales

Relationships include

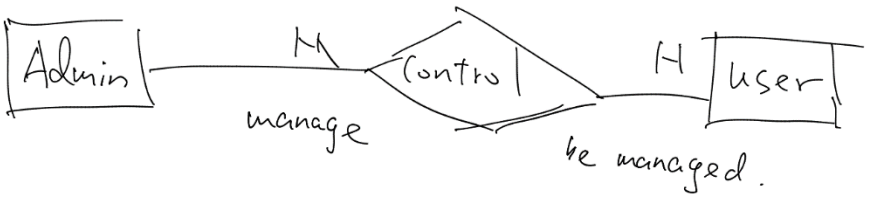




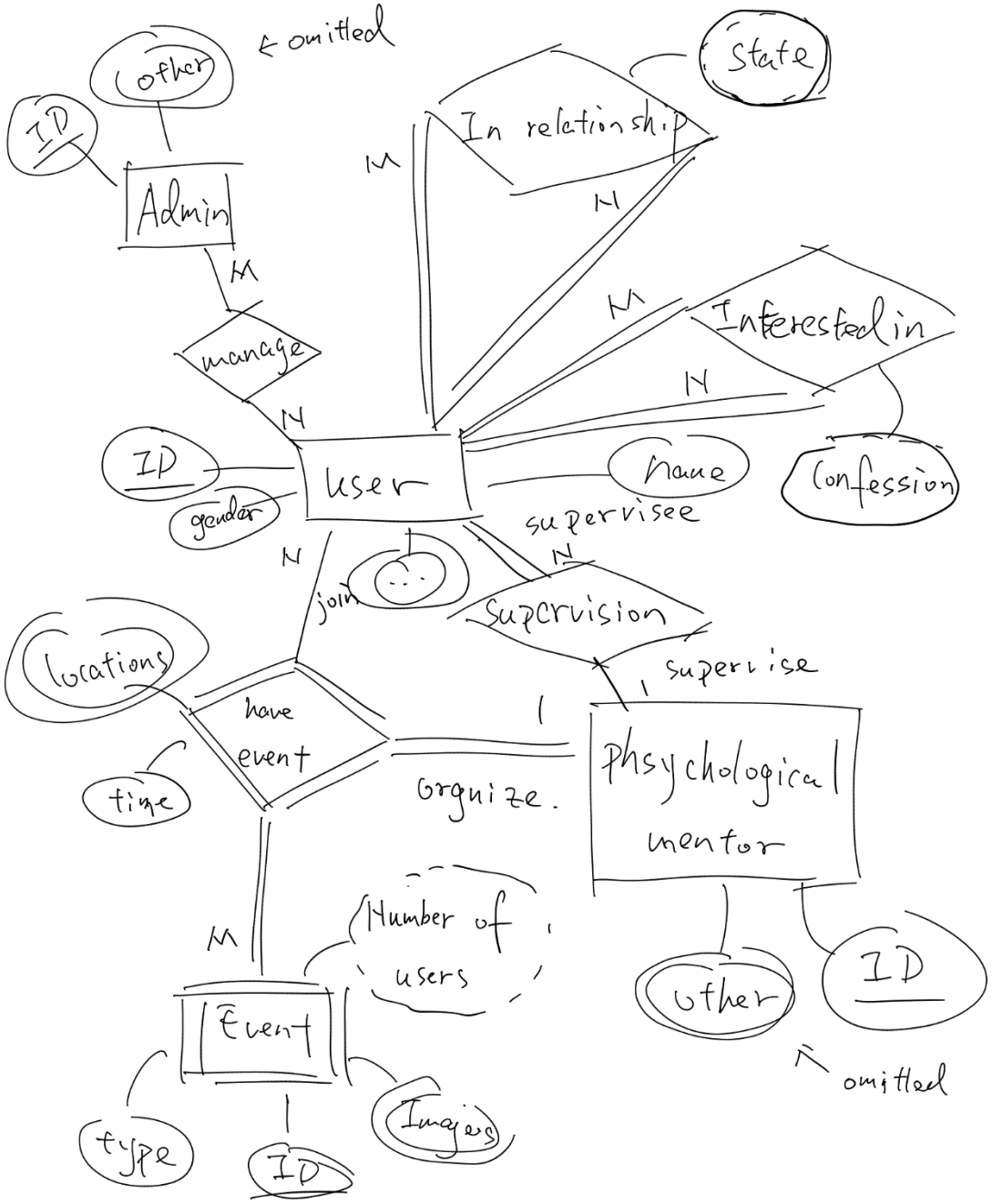
Can have event together,

they can have no more than

one mentor to go with them.



And the rough total relationship would be

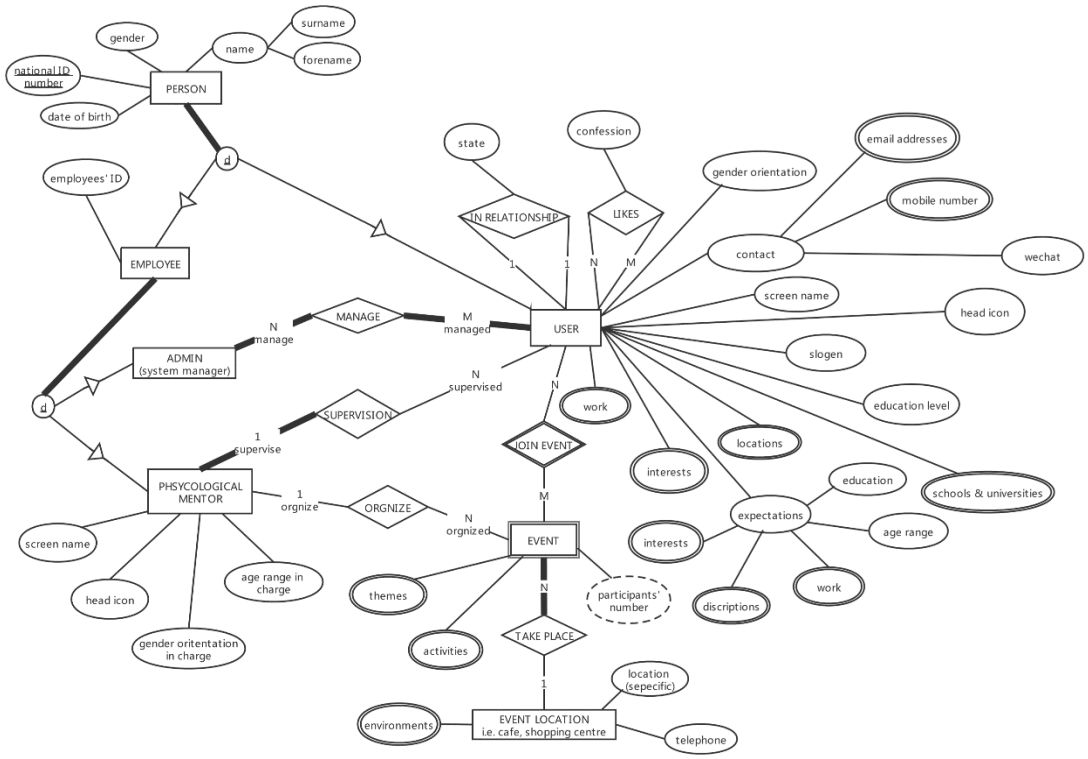


5-21

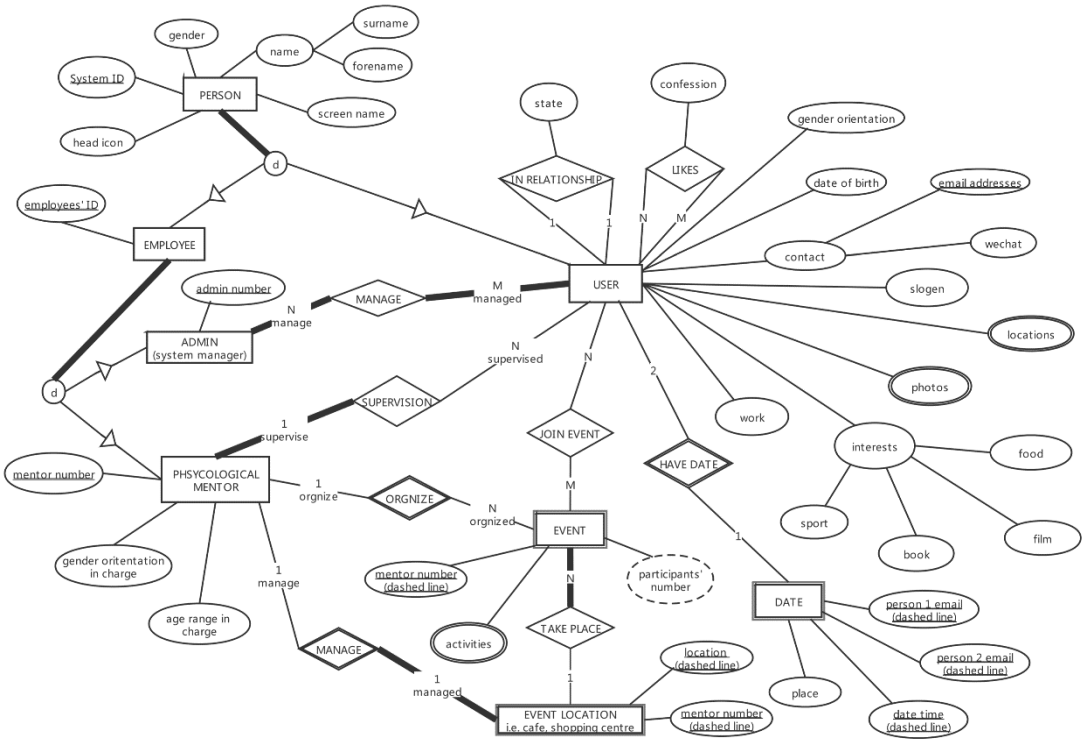
This week we created our first version ER diagram of our data system, and we improved it, eliminated it and get out second version. Our database will be mapped based on this second

version ER diagram.

First version



Second version

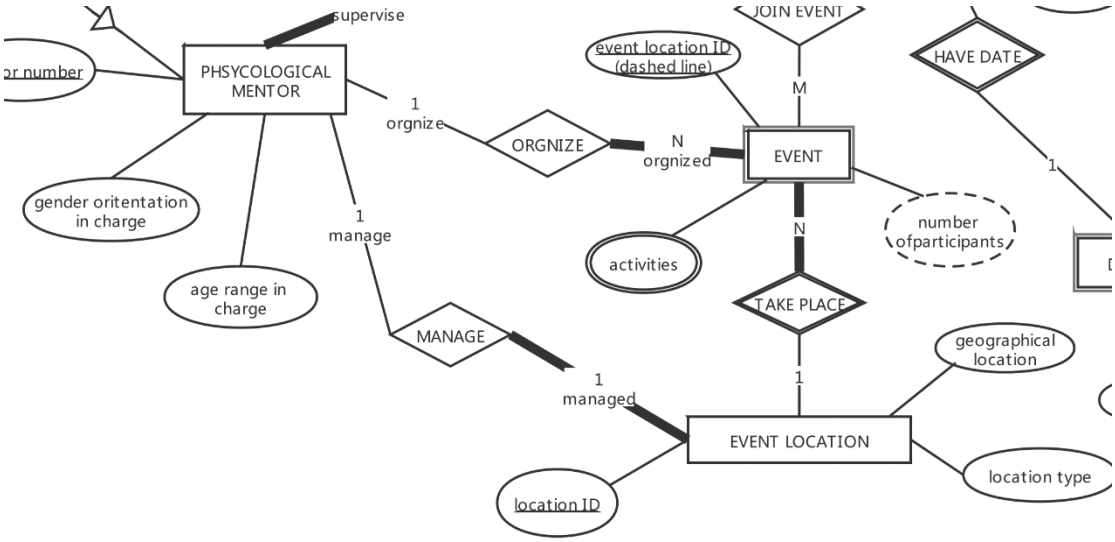


In the first version we designed hierarchical structure of three types of person as entities, which are system administrators, psychological mentors and users. Among which, admins and mentor

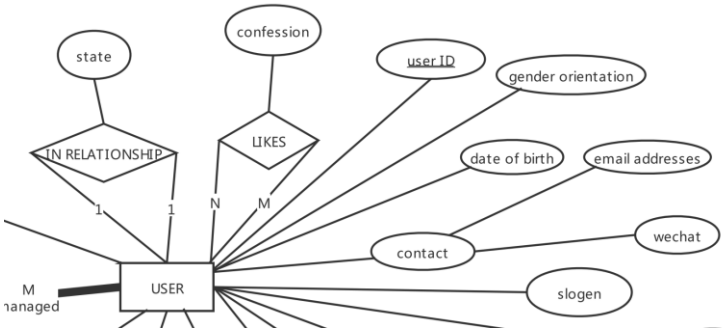
are employees of this system, and the three of them all belong to person entity. We also prepared an event entity which is a weak entity depends on user.

In the second version we simplified some attributes from user entity; we also changed a weak entity - event to make it depend on mentor, and decided to use mentor ID as its primary key; similarly, we also designed an event place entity which is also weak and depend on mentors; In addition, we add a date weak entity that uses users to be its primary key.

In the third version we simplified this event and event location system, redesigned some attributes and change the event attribute to depend on location, not mentor anymore. And since a location can be more static than a person – mentor, we changed it to be a normal entity not a weak one.



And in the fourth edition we add user ID back and use this as the primary key, not email address anymore.



We will map our database to this third version of ER diagram. And we will do this following the algorithm introduced in the lecture, except that we will consider sub-entities at the very beginning.

12. Mapping regular entity types (and consider the sub-entities)

PERSON

System ID	Surname	Forename	Gender	Screen name	Head icon

EMPLOYEE

System ID	Employee ID

ADMINISTRATOR

System ID	Employee ID	Admin number

PHSYCOLOGICAL MENTOR

System ID	Employee ID	Mentor number

Gender orientation in charge	Age range in charge

USER

System ID	User ID	Email address	Wechat	Gender orientation	Date of birth

Slogan	Locations (multiple values)	Photos (multiple values)	Work	Food	Film

Book	Sport

EVENT LOCATION

Location ID	Location type	Geographical location

13. Mapping weak entities

EVENT

Location ID	Time HHDDMMYY	Number of participants	Activities (multiple values)

DAT E

Uid1	Uid2	HHDDMMYY	Geographical location

14. Mapping 1: 1 relationship

MANAGE (some mentors manage location)
In EVENT LOCATION

Location ID	Location type	Geographical location	Manager ID	Start time DDMMYY

IN RELATIONSHIP and HAVE DATE

In USER

System ID	User ID	Email address	Wechat	Gender orientation	Date of birth

Slogan	Locations (multiple values)	Photos (multiple values)	Work	Food	Film

Book	Sport	Relation Uid	Relation state

15. Mapping 1: N relationships

SUPERVISION

In USER

System ID	User ID	Email address	Wechat	Gender orientation	Date of birth

Slogan	Locations (multiple values)	Photos (multiple values)	Work	Food	Film

Book	Sport	Relation Uid	Relation state	MENTOR ID

ORGNIZE

In EVENT

Location ID	Time HHDDMMYY	Number of participants (calculated through relationship)	Activities (multiple values)	MENTOR ID

16. Mapping M: N relationship

JOIN EVENT

Uid	EVENT LOCATION ID	TIME HHDDMMYY

LIKES

Uid1	Uid2	Confession

17. Mapping multivalued attributes

USER locations will be written as a string in designed format, so as EVENT activities.

USER PHOTOS

Uid	Photo path (in the system)

18. Mapping N-ary relationships

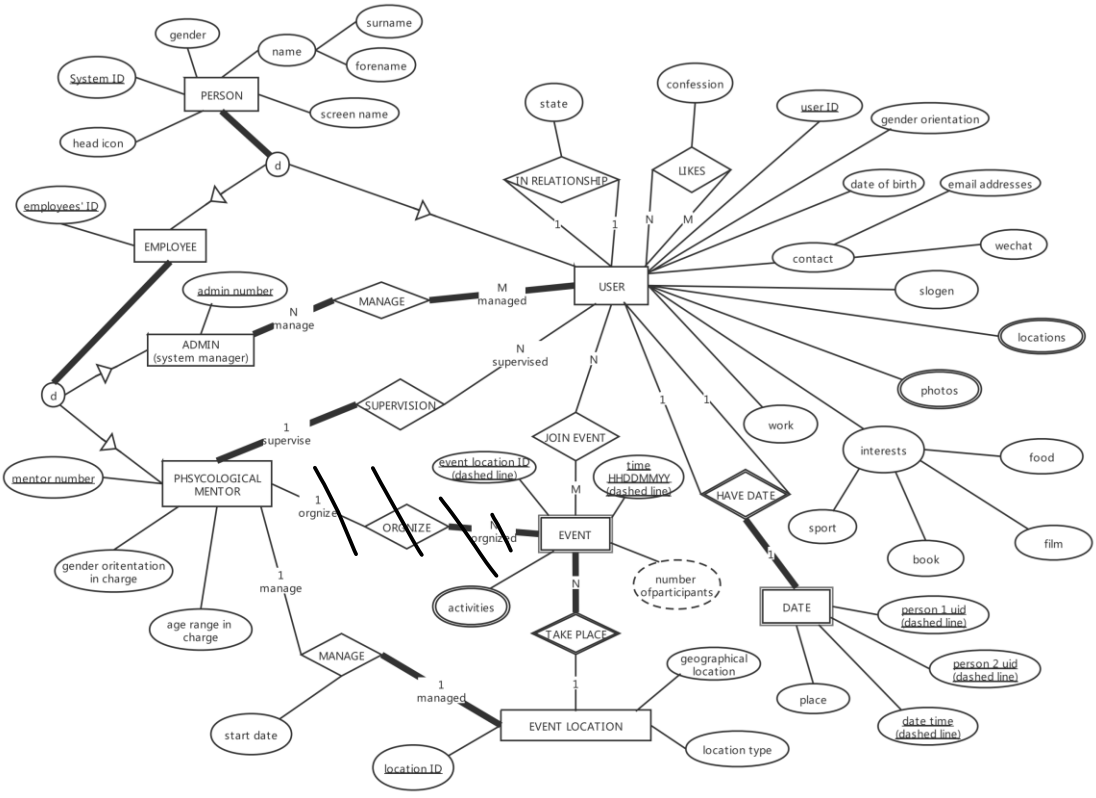
We've already cancelled all the relationships that have more than 2 types of entity referred, so we can skip this step.

19. Mapping supertype and subtypes

We've done this above; we keep the primary key of the supertype as a foreign key in the subtype.

5-27

During the period we've designed our last version of ER diagram and we try to keep this as our final version (though there might be further modification).



With this diagram we would give our final design of database tables and we will give our normalized version of it later.

PERSON

System ID	Surname	Forename	Gender	Screen name	Head icon	password
Auto increment integer	Varchar(64)	Varchar(64)	Enum('female', 'male', 'unselected')	Char(64) Unique	Varchar(128) (system paths for images)	Char(64) Not null

EMPLOYEE

System ID	Employee ID
Foreign key from PERSON	Auto increment integer

ADMINISTRATOR

System ID	Employee ID	Admin number
Foreign key from PERSON	Foreign key from EMPLOYEE	Auto increment integer

PHSYCOLOGICAL MENTOR

System ID	Employee ID	Mentor number
Foreign	Foreign	Auto inc int

Gender orientation in charge	Age range in charge begin	Age Range end
Enum('homosexual', 'hetero')	int	int

USER

System ID	User ID	Email address	Wechat	Gender orientation	Date of birth
Foreign	Auto inc int	Varchar(128)	Varchar(64)	Enum('homosexual', 'hetero') default 'hetero'	DATE

Slogan	*Locations (multiple values)	*Work	*Food	*Film
Varchar(512)	Char(30)	Char(9)	Char(30)	Char(30)

*these are multiple value attributes in design, but in practical they store labels with maximum length of 3 (e.g., '02;'), so they are just chars in the DB.

*Book	*Sport	Relation Uid	Relation state	MENTOR ID
Char(30)	Char(30)	Foreign	Enum('acquaintance', 'ambiguous', 'boyfriend/girlfriend', 'engaged', 'married')	Foreign key from MENTOR

Relation Uid and Relation state are removed later.

EVENT LOCATION

Location ID	Location type	Geographical location	Manager ID	Start time DDMMYY
Auto inc int	Enum('coffee /tea house', 'bar', 'restaurant', 'shopping centre', 'part', 'cinema', 'theatre')	Varchar(256)	Foreign	DATE

EVENT

Location ID	Time HHDDMMYY	Number of participants (calculated through relationship)	Activities (multiple values)
Foreign	DATETIME	Integer	Char(15)

The mentor here may not be the same person that manages this location

DAT E

Uid1	Uid2	HHDDMMYY	Geographical location
Foreign	Foreign	DATETIME	Varchar(256)

JOIN EVENT

Uid	EVENT LOCATION ID
Foreign	Foreign

LIKES

Uid1	Uid2	Confession
Foreign	Foreign	Varchar(512)

USER PHOTOS

Uid	Photo path (in the system)
Foreign	Varchar(128)

*Foreign represents foreign key from USER.

LABELS

Serial	Locations	Work	Food	Film	Book	Sport	Activity
Unique auto inc	Varchar(64)	--	--	--	--	--	--

i.e.

1	Chengdu	IT/CS	Spaghetti	Titanic	Jane Eyre	Tennis	Blind date
2	NewYork	Education	--	--	--	--	--
3	Beijing	Sales	--	--	--	--	--
4	Shanghai	Finance	--	--	--	--	--

The next step is to turn these data design into normal forms.

The 1st normal form requires no repeating attributes. At present level of table design, we actually have one repeating attributes, which is the "state" of relationships. Although these should be 1-to-1 relationships, this attribute of "state" is actually kept in both users, but they are exactly same, so they are stored twice, and they need double modifications. According to the course slides, this can be a case of having a repeating group. To normalize this, we can add an extra table for this relationship, just like what we've done in "LIKES".

So, it should be:
RELATIONSHIP

Uid1	Uid2	Relation state
Foreign USER key	Foreign USER key	Enum('acquaintance', 'ambiguous', 'boyfriend/girlfriend', 'engaged', 'married')

Where we keep the second uid with greater value than the first; and we can remove these attributes from USER.

For 2NF, it is defined that, a relation is in 2NF if, and only if, it is in 1NF and every non-key attribute is fully functionally dependent on the whole key. For our case, we have several tables with more than one primary keys, and for them, we need to check each of the attributes, to see if they depend on all the primary keys, if not, we would take them out of the original table.

Our tables with more than one primary key are EVENT, DATE, JOIN EVENT, and LIKES. However, we checked them one by one and we think all the attributes in them must depend on all the primary keys. We've already considered these issues while we were designing our ER diagram and tables.

For 3NF, we searched for transitive functional dependencies. The only thing we found that might refer to this issue is that in EVENT LOCATION, it feels like the start time of a mentor's management should depend on this mentor. But consider a case that a mentor originally manages a place, and now he/she goes on a vacation, someone else take this job for a period, but then he/she returns, and starts to manage again. This example shows that the time is actually

independent of mentor, so we don't need change anything for it. As a result, we are already in 3NF.

For BCNF, we need to solve issue of overlapping candidate keys. In our system, using USER table as an example, there are actually two determinant keys, user ID and system ID. But we already solve the issue of this overlapping dependency, by keeping user ID as the primary key, and letting all the local attributes depend on the user ID, not system ID. In addition, as it is shown in the ER diagram and the tables above, we've made a classification of attributes, for PERSON, EMPLOYEE, ADMIN, MENTOR and USER. They all have this determinant key issue originally, but we separate the shared attributes to the higher level entities, and cutoff such shared dependency, only keep the candidate keys as foreign keys (not primary keys). As a result, we may say that we are already in BCNF.

Design of webpages and functions

3. Login page

Allow login process for all the persons uses their screen name (unique) and password.

Two input bars
- User name
- Password

Two buttons
- Login
- Register now

Response: e.g., "username does not exist", "wrong password".

Login – as we've done, it jumps to login servlet; with request message with username (screen) and password.
Register – turns to users' registration page.

4. Users' registration page

For adding user's information in PERSON table.

Input bars
- Surname
- Forename
- Gender

Button
- Next – send the three pieces of information as request message and jump to 3. Users' registration page (2).

5. Users' registration page (2)

User's information in PERSON table, continues

Input bars

- Username
- Password

Button

- Next – send the two pieces of information as request message and jump to user system registration servlet.
- Last – turn back to last page

Response info: "username already exist, please change one!", "cannot enter empty string!"

6. Users' attributes page

Input bars

Contact:

- Email
- Wechat

- Gender orientation ('hetero', 'homosexual')
- Date of birth select (year, month, day)

Character:

- Slogan

In addition to slogan, the following items are all selected from a list of tags, as shown in the figure on the right. Here I will give each tag a list of corresponding numbers and words. For example, in locations, 1-beijing, 2-Shangai and 3-tianjin are displayed. Press one button to add a number to a string, For example, if I choose Beijing and Shanghai, the string given to me should be "1; 3;".

We give each item a row of buttons and write down the number of buttons.

It may be the most difficult web page, but if it can be realized, it will definitely be the highlight.

Locations

- Work
- Food

- Films
- Books
- Sports

Button

- Ok – send the two pieces of information as request message and jump to user table attributes servlet."

7. Upload head icon page

Input bar:

- Select file

Button:

- Submit, give response with image file to head icon servlet.

Response info: "please upload .jpg/.png files"

Refer to this:
https://www.runoob.com/servlet/servlet-file-uploading.html

This page doesn't show the uploaded image, but leaves space. It doesn't matter about it appears or not, we just need the structure.

https://www.codejava.net/coding/upload-files-to-database-servlet-jsp-mysql

8. User's main page

These buttons are static:
Log out, change my characters, change my head icon, change my information;

Four sub-pages:
Recommendations, events, likes, my mentor;

1. Recommendations

Head icon	Screenname	Gender	Slogan	Locations	Work	Food	Films	Sports	Books
--	MBT	Male	Heihei	BJ; Chongqing; …	IT; student; …	--	--	--	--
--	--	--	--	--	--	--	--	--	--
--	--	--	--	--	--	--	--	--	--

2. Events

Location type	Activities	Time	Location	Participants number	Mentor screenname

Tea house	Blind date	--	--	10	BTM

3. Likes

Who likes me? I like whom?

Head icon	Screenname	Gender	Slogan		Head icon	Screenname	Gender	Slogan
--	MBT	Male	Heihei		--	MBT	Male	Heihei
--	--	--	--		--	--	--	--
--	--	--	--		--	--	--	--

4. My mentor

If "I" don't have mentor yet, "I" can choose from the list, if "I" have, "I" won't see this list.

Head icon	Screen name	Gender	Age
--	ZMX	Female	18

On these for pages, if clicking mouse on each row, it will jump to different pages.

6-2

Redesigned User Characteristics

In the last version we used fixed strings (char) to record labels of users' likes such as food and sports. However, we now redesigned these into several M to N relationships, so that we can store infinite number of labels for each user, and it's easier for us to search persons with same labels.

USER

System ID	User ID	Email address	Wechat	Gender orientation	Date of birth
Foreign	Auto inc int	Varchar(20)	Varchar(20)	Enum('homosexual', 'hetero') default 'hetero'	DATE

Slogan	Work	MENTOR ID
Varchar(20)	One number from Labels	Foreign key from MENTOR

Location

Lid	Uid

Food

Fid	Uid

Books

Bid	Uid

Films

Fid	Uid

Sports

Sid	Uid

More pages

Admin Page

Subpage 1.
Top button: “add user”
On each line - Show all the users’ system id, user id, username, and forename + surname and gender
Two buttons “modify” and “remove”;
Each page – 20 users; next page; last page; search userID: _____ [submit]
Click “modify” to a modify page (can only modify info in Person, including password, but not User).
Click “remove” to a remove page.
Click “add user” to an add user page.

Modify page:
Can change user id, username, forename, surname, gender, password; but not system ID.

Delete page:
“Are you sure to delete user <systemID>? You may not undo this!”
Password: ____
Button: “confirm”

Add user page:
99% same as user’s registration page.

Subpage 2
Manage mentors, same functions

Mentor’s Page

Subpage 1 users
Can only view the user’s that supervised by him/her.

But more information:

Head icon	Screenname	Gender	Slogan	Locations	Work	Food	Films	Sports	Books

And also, **wechat** and **email**.

Each page 20 users, can **search username** (screen name).

Subpage 2 event location

If having one location managed, show location info:

Location ID	Location type	Geographical location

[add location (only appears when there's no location)] [delete]

Delete page:

"Are you sure to delete user <Location ID>? You may not undo this!"

Password: ____

Button: "confirm"

And is able to view the events under this location:

EventID, time, activities, number of participants [remove]

[add event] [next page] [last page] [search ID]

Add event page:

Time, activities, submit;

Activities (multiple choise)

According to the above, established the database.

```
+----------------------+
| Tables_in_groupwork  |
+----------------------+
| administrator        |
| books                |
| date                 |
| employee             |
| event                |
| event_location       |
| films                |
| food                 |
| join_event           |
| labels               |
| likes                |
| location             |
| person               |
| phsycological_mentor |
| relationship         |
| sports               |
| user                 |
| user_photos          |
+----------------------+
18 rows in set (0.02 sec)
```

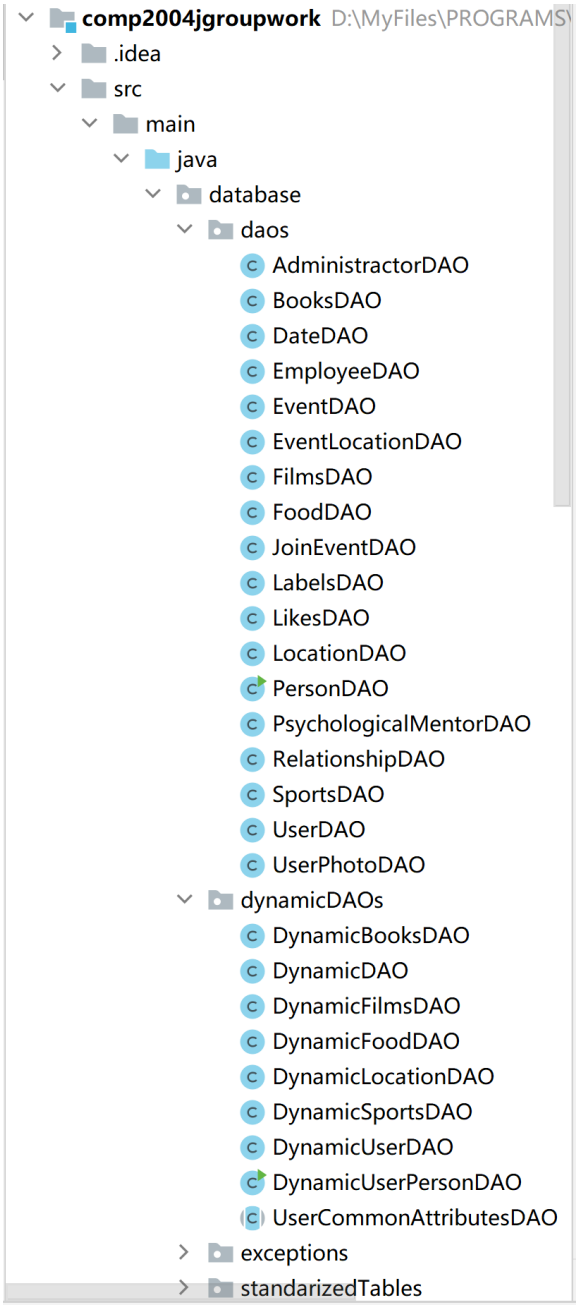
All the files in daos used Hibernate ORM, which explained in Week13. Hibernate helps us persistently map into database tables, reducing the hassle of extracting information directly from the tables. From every table's DAO, it has 5 basic methods: get all rows from every table/ save one object to the table/ get one object by key/ delete object by key/ update one object.

Compared with use sql method with Connection and Statement, using Hibernate did not need to call session every time. It saves a lot of time, especially when it needs to check a lot of data through one table. Beyond that, using Hibernate brings convenience for us to write unity. For inserts and updates, Hibernate provides more freedom for writing, which Hibernate can directly return a created object.

```
public static List<Sports> getAllSports() {
    List<Sports> l  = null;
    try (Session session = HibernateUtil.getSessionFactory().openSession()) {
        l = session.createQuery( s: "from Sports").list();
        session.close();
    } catch (Exception e) {
       e.printStackTrace();
    }
    return l;
}

public static void saveSports(Sports e) {
    Transaction transaction = null;
    try (Session session = HibernateUtil.getSessionFactory().openSession()) {
        transaction = session.beginTransaction();
        session.save(e);
        transaction.commit();
    } catch (Exception exp) {
        if (transaction != null) {
            transaction.rollback();
        }
        exp.printStackTrace();
    }
}

public static Sports getSportsByKey(int sid,int uid) {
    Sports l = null;
    try (Session session = HibernateUtil.getSessionFactory().openSession()) {
        l = (Sports)session.createQuery( s: "from Sports where Sid = "+ sid + " and Uid = "+uid).uniqueResult();
        session.close();
    } catch (Exception e) {
       e.printStackTrace();
    }
    return l;
```

The files in dynamicDAOs are re-designed version of DAOs above, which need one instance to operate database. It is not static, and the sessions are only closed after finishing all the operations needed, which saves a lot of time for opening and closing sessions. It is worth mentioning that the method used in DynamicUserPerson is from Week6 lecture, sql statement in java. The reason is that in this function, we wanted to join the user table and person together but, hibernate do not support sql JOIN method. In this function, we used prepared statement as mentioned in the lecture.

```java
public UserPerson getUserPersonBySystemID(int systemID) {
    try {
        String columns = "person.SystemID, person.Surname, person.Forename, person.Gender, person.ScreenName," +
                "person.HeadIcon, person.password, user.UserID, user.Emailaddress, user.Wechat, user.GenderOrientation, " +
                "user.DataOfBirth, user.Slogan, user.Work, user.MentorID";
        String sql = "SELECT " + columns + " FROM user inner join person on user.SystemID = person.SystemID";
        sql = "SELECT * FROM (" + sql + ") as UserPerson WHERE SystemID = ?;";
        PreparedStatement ps = connection.prepareStatement(sql);
        ps.setInt( parameterIndex: 1, systemID);
        System.out.println(ps);
        ResultSet rs = ps.executeQuery();
        rs.next();

        int userID = rs.getInt( columnLabel: "UserID");
        String emailAddress = rs.getString( columnLabel: "Emailaddress");
        String wechat = rs.getString( columnLabel: "Wechat");
        String genderOrientation = rs.getString( columnLabel: "GenderOrientation");
        Date dataOfBirth = rs.getDate( columnLabel: "DataOfBirth");
        String slogan = rs.getString( columnLabel: "Slogan");
        int work = rs.getInt( columnLabel: "Work");
        int mentorID = rs.getInt( columnLabel: "MentorID");
        String surname = rs.getString( columnLabel: "Surname");
        String forename = rs.getString( columnLabel: "Forename");
        String gender = rs.getString( columnLabel: "Gender");
        String screenName = rs.getString( columnLabel: "ScreenName");
        String headIcon = rs.getString( columnLabel: "HeadIcon");
        String password = rs.getString( columnLabel: "password");

        ps.close();
        UserPerson userPerson = new UserPerson(systemID, userID, emailAddress, wechat, genderOrientation, dataOfBirth,
                slogan, work, mentorID, surname, forename, gender, screenName, headIcon, password);
        return userPerson;
    } catch (SQLException e) {
```



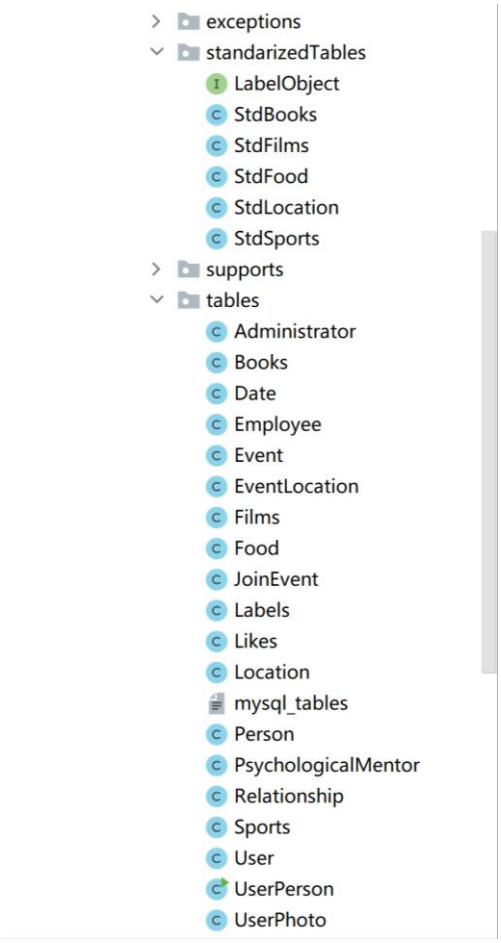
We created one interface LabelObject for all interests, like Sports, Books, Films, Food and traveled location. These table have one common characteristic that they only include their label ID and user ID and they takes two together as their primary key. With this inerface, many method of different classes can be written into one.

```java
package database.standarizedTables;

public interface LabelObject {
    public int getLabelId();
    public void setLabelId(int labelId);
    public int getUserId();
    public void setUserId(int UserId);
}
```

```java
DynamicUserPersonDAO userDAO = new DynamicUserPersonDAO();
UserCommonAttributesDAO commonAttributesDAO;
List<CompareNode> remain = null;
for (String attributeName: LABELS) {
    switch (attributeName) {
        case "StdBooks":
            commonAttributesDAO = new DynamicBooksDAO();
            break;
        case "StdFilms":
            commonAttributesDAO = new DynamicFilmsDAO();
            break;
        case "StdFood":
            commonAttributesDAO = new DynamicFoodDAO();
            break;
        case "StdLocation":
            commonAttributesDAO = new DynamicLocationDAO();
            break;
        case "StdSports":
            commonAttributesDAO = new DynamicSportsDAO();
            break;
        default:
            throw new WrongAttributeNameException("Attribute " + attributeName + " is not valid");
    }

    List<CompareNode> keepList = matchForOneTypeOfLabel(userPerson, commonAttributesDAO, userDAO,
            genderPreference, targetGenderOrientation);
    if (remain != null) {
        // merge nodes from two lists
        List<CompareNode>[] lists = CompareNode.mergeNodes(remain, keepList);
        remain = lists[0]; keepList = lists[1];
```

To control the objects successfully, we created Stdxxxx for every interested which implement LabelObject interface and use Hibernate interface with the orignal table. In every Std class, it could get every LabelID anf UserID, which is uniform and easy to manage.

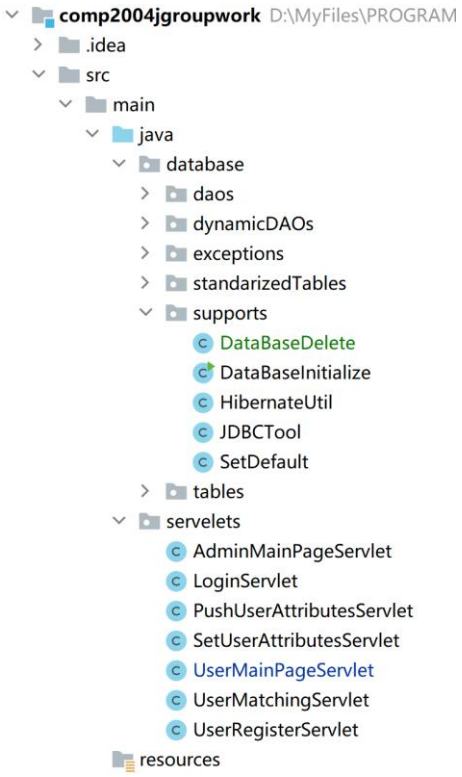
```java
@Override
public int getLabelId() { return Sid; }

@Override
public void setLabelId(int labelId) { this.Sid = labelId; }

@Override
public int getUserID() { return Uid; }

@Override
public void setUserId(int userId) { this.Uid = userId; }
```

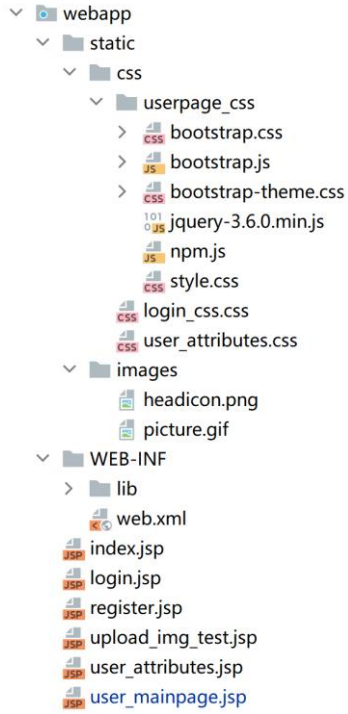


In the subsequent process, we found that this created some inevitable problems. Such as when we want to get mutiple object though one judgement condition. e.g. find all users' ID which interested sport's ID = 5. It will return a lot of IDs but is repetitive. We checked a lot of time, and the grammar and logic is correct. After checking on the Internet, we know that if use multiple interfaces of Hibernate mapping, it will only displayed the first data queried repeatedly. After several fruitless modifications, we decided to go back to sql method with Connection and Statement. And then we got a hopeful answer.

```
@Override
public List<LabelObject> getAllValuesWithUserID(int userID) {
    List<LabelObject> list = new LinkedList<>();
    try {
        PreparedStatement ps = connection.prepareStatement( sql: "SELECT * FROM sports WHERE Uid = ?;");
        ps.setInt( parameterIndex: 1, userID);
        ResultSet rs = ps.executeQuery();
        while (rs.next()){
            int Sid = rs.getInt( columnLabel: "Sid");
            int Uid = rs.getInt( columnLabel: "Uid");
            StdSports s = new StdSports(Sid,Uid);
            list.add(s);
        }
        rs.close();
        ps.close();
    } catch (Exception e) {
        e.printStackTrace();
        list = null;
    }
    // for (LabelObject each: list) System.out.println((each.getLabelId()));
    return list;
}
```

```java
@Override
public List<LabelObject> getAllValuesWithLabelID(int labelID) {
    List<LabelObject> list = new LinkedList<>();
    try {
        PreparedStatement ps = connection.prepareStatement( sql: "SELECT * FROM sports WHERE Sid = ?;");
        ps.setInt( parameterIndex: 1, labelID);
        ResultSet rs = ps.executeQuery();
        while (rs.next()){
            int Sid = rs.getInt( columnLabel: "Sid");
            int Uid = rs.getInt( columnLabel: "Uid");
            StdSports s = new StdSports(Sid,Uid);
            list.add(s);
        }
        rs.close();
        ps.close();
    } catch (Exception e) {
        e.printStackTrace();
        list = null;
    }
    for (LabelObject each: list) System.out.println((each.getUserID()));
    return list;
}
```




webapp
static
css
userpage_css
bootstrap.css
bootstrap.js
bootstrap-theme.css
jquery-3.6.0.min.js
npm.js
style.css
login_css.css
user_attributes.css
images
headicon.png
picture.gif
WEB-INF
lib
web.xml
index.jsp
login.jsp
register.jsp
upload_img_test.jsp
user_attributes.jsp
user_mainpage.jsp