Xi Cheng (999872475)
Jun Ma (912381574)

ECS 165A

Homework 4 Write Up

1.
We have created four tables for the university database for FakeU.

```
Student (SID, SURNAME, PREFNAME, EMAIL)
Take (SID, CID, TERM, SEAT, GRADE, STATUS, MAJOR, UNITS_RECIEVED, GPA)
Course (CID, TERM, SUBJ, CRSE, SEC, UNITS)
Meetings (CID, TERM, DAYS, TIME, BUILD, ROOM, INSTRUCTOR, TYPE, STARTTIME,
ENDTIME, DAY)
```

- Student

The Student table contains the basic information of each unique student who took courses in any terms from summer 1989 to summer 2012. The student ID (SID) is unique for all students, and can determine any other information of a single student.
We would store SID as INTEGER, store SURNAME, PREFNAME, and EMAIL as VARCHAR.
We have functional dependencies as followed.

```
SID -> SURNAME, PREFNAME, EMAIL
EMAIL -> SURNAME, PREFNAME, SID
```

- Take

The Take table presents the relationship between each student and the course he or she took. Therefore, student ID, course ID, and Term would be the key. We could use them to determine other attributes and link a specific student to a course. Also, the GPA attribute is a transformation from GRADES in alphabets to points and it's a more straightforward way to calculate the cumulative grades for letter grading.
We would store SID, CID, SEAT, and UNITS_RECIEVED as INTEGER, store GRADE, TERM, LEVEL, CLASS, STATUS, MAJOR as CHAR, and store GPA as NUMERIC(2,1).
We have functional dependencies as followed.

```
SID, CID, TERM -> SEAT, GRADE, STATUS, MAJOR, UNITS_RECIEVED, GPA
```

- Course

The Course table contains the information of each course provided in any terms from summer 1989 to summer 2012. Since the course ID (CID) could be reused in different terms, we need to have both CID and term as a key to determine a single course.
We could store CID, SEC, and CRSE as INTEGER, store TERM, SUBJ and UNITS as VARCHAR/CHAR.
We have functional dependencies as followed.

```
CID, TERM -> SUBJ, CRSE, SEC, UNITS
SUBJ, CRSE, SEC, TERM -> CID, UNITS
```

- Meetings

The Meetings table contains the detailed information of different meetings in a course, so it needs the key of Course. Also, we could determine a single meeting by days, time, building, and room, because no two meetings would take place in the same location at the same time. We add the attributes STARTTIME, ENDTIME, and DAY to identify courses that have conflicts due to summer sessions merging.
We could store CID, TERM, and ROOM as INTEGER, and store others as VARCHAR(50).
We have functional dependencies as followed.

```
CID, TERM, DAYS, DAY, TIME, BUILD, ROOM -> INSTRUCTOR, TYPE, STARTTIME,
ENDTIME
CID, TERM, DAY, TIME, BUILD, ROOM -> INSTRUCTOR, TYPE, STARTTIME, ENDTIME, DAY
TIME -> STARTTIME, STARTTIME
```

The only MVD we expect to hold is SID ->-> CID for the Take table, since we could find a tuple s in the take relation that has SID and CID agree with a random tuple t, and other attributes that agree with another tuple v. As for the other relations, we do not expect any MVD for them to hold because the student relation has unique values of SID which cannot be repeated, and similar reasoning goes for the other relations

- For future updates:

-- *Updating*: If we want to update the basic information of a student, such as preferred name, etc., we could update the Student table using the existing student ID. If we want to update the information of a course, such as section, units, or course number, etc., we could update the Course table using the existing course ID and term. If we want to update the information of some meetings of a course, such as instructor, or time, etc., then we could update the Meetings table using the key of the table. If we want to update a relationship, such as seat, grade, or status, etc., we could update the Take table using the known student ID, course ID, and term.
-- *Inserting/Deleting*: If we want to add new courses, we could add tuples in Course table and Meetings table using the new information, as well as the Take table with null value for attributes related to students. If we want to add new students to a course, we could update the Take table with the new information (level, class, etc.) of the student, or add new tuples to Student table if the student is the first time to take course. Also, we could have the same procedure for deleting.

2.
For loading the grade data, we created *start_table.py* to create table and *load_data.py* to load all data to the tables, with more detailed directions in the readme file. The version of Python we used is 2.7.12.

3.
All codes doing the calculation in this question were written in *query.py*.

a) We used the following methods:

- Found numbers of students for each term, sum them up, so we could get the total number of students who have taken courses
- For each term, found the numbers of students who attempt 1, 2, …, 20 units
- Sum up the numbers of students who take 1, 2, …, 20 units in each term, divide each of these 20 results by the total number of students who have taken course

The results are shown below.

The number of unit students attempted	**Percentage**
1	1.579015%
2	0.411569%
3	1.668695%
4	45.219717%
5	3.031516%
6	1.287554%
7	1.976171%
8	12.620108%
9	3.648069%
10	1.551790%
11	1.992185%
12	13.922074%
13	5.593011%
14	1.826436%
15	0.988085%
16	1.368426%
17	0.735859%
18	0.157741%
19	0.089680%
20	0.054449%

b) We used the following method:

- For each term, calculate the numbers of students, their sum weighted GPAs, and the total units they attempted for each group of students.
- For each group of students, add their sum weighted GPAs and their total units, use the formula *(GPA of a course) * (the unit of that course) / (total units)* to calculate the average GPA.

The results are shown below.

The number of unit students attempted	**Average GPA they got**
1	3.515000
2	3.663504
3	3.404276
4	2.737998
5	2.668381

6	3.463715
7	3.086003
8	2.885982
9	2.714298
10	3.058307
11	3.009342
12	2.886233
13	2.717674
14	2.712079
15	2.931360
16	2.866838
17	2.868916
18	2.688801
19	3.232211
20	2.664697

c) We used the following methods: for each instructor, calculate the sum GPAs and total units their students got, then use the formula *(GPA of the course) * (the unit of that course) / (total units)* to calculate the average GPA.
After doing the query, we could find that the hardest instructor is Emily A. Turner, and the average grade she assigned is 1.70. The easiest instructors are Madison G. O'donnell and Angel J. Russo, and the average grade they assigned is 3.95.

d) We divided the courses into two groups, letter grading courses and P/NP courses.

For letter grading courses:

Course	**The most difficult instructor(s)**	**Average grade of the most difficult**	**The easiest instructor(s)**	**Average grade of the easiest**
ABC 101	Diaz, Riley I.	2.151250	Baldwin, Ella J.	3.151362
ABC 102	Parsons, Mia E.	3.160870	Parsons, Mia E.	3.160870
ABC 103	Williams, Victoria M.	2.640741	Donaldson, Matthew C.	3.460000
ABC 104	Miller, Emma J.	1.955403	Murphy, Melanie S.	3.204082
ABC 105	Adams, Emily G.	1.823256	Williams, Victoria M.	3.308060
ABC 106	Whitehead, William A.	1.487805	Dodson, Nicole M.	3.462802
ABC 107	Bates, Logan Q.	2.372507	Edwards, Maya N.	3.107910
ABC 108	Green, Isabella M.	1.805658	Olson, Jennifer D.	2.984461
ABC 109	Fisher, Caleb K.	3.007232	Parsons, Mia E.	3.139855
ABC 110	Cobb, Sophie A.	3.542500	Cobb, Sophie A.	3.542500

ABC 111	Morris, Evan E.	3.957143	Morris, Evan E.	3.957143

For P/NP courses: (we ignore the null/unknown)

Course	The most difficult instructor(s)	Pass rate	The easiest instructor(s)	Pass rate
ABC 112	Perry, Katherine V. Cox, Ayden C. Morris, Evan E. Williams, Victoria M. Olson, Jennifer D. Fisher, Caleb K.	100%	Perry, Katherine V. Cox, Ayden C. Morris, Evan E. Williams, Victoria M. Olson, Jennifer D. Fisher, Caleb K.	100%
ABC 113	Diaz, Michelle H.	86.7%	Herring, Nathan L. White, Sophia V. Morris, Evan E.	100%
ABC 114	Oliver, Sebastian I.	0%	Diaz, Michelle H. Battle, Gianna M. Rutledge, Ashley B. Cox, Jayden L. Morris, Evan E. Williams, Victoria M. Downs, Jesus C. Cox, Ayden C. Olson, Jennifer D. Williams, Juan L. Sullivan, Jordan H.	100%

e) We use the methods from piazza @923.
The courses that have meeting conflicts are listed below.

Term	Subj	Course	Subj	Course
199106	ABC	105	ABC	106
199406	ABC	107	ABC	105
199406	ABC	107	ABC	106
199506	ABC	107	ABC	104
199506	ABC	107	ABC	105
199606	DEF	201	DEF	258
199706	ABC	104	ABC	106
199706	ABC	104	DEF	201
199706	ABC	106	ABC	105
199906	ABC	106	ABC	105
199906	DEF	201	ABC	104
200006	ABC	108	ABC	104
200106	ABC	108	ABC	104
200106	DEF	201	ABC	105
200106	DEF	201	ABC	105
200406	ABC	106	ABC	105
200406	DEF	229	DEF	258

200606	ABC	105	DEF	250
200606	DEF	201	DEF	258
200606	DEF	201	DEF	258
200606	DEF	201	DEF	258
200606	DEF	258	DEF	201
200706	DEF	214	ABC	106
200706	DEF	258	DEF	238
200706	DEF	258	DEF	238
200806	ABC	104	ABC	101
200806	ABC	104	DEF	201
200806	DEF	201	DEF	258
200806	DEF	214	DEF	258
200806	DEF	238	DEF	258
200806	DEF	250	DEF	258
200906	ABC	104	ABC	107
200906	ABC	104	ABC	221
200906	ABC	201	ABC	221
200906	DEF	103	ABC	104
200906	DEF	201	ABC	108
200906	DEF	201	DEF	258
200906	DEF	201	ABC	108
200906	DEF	293	DEF	292
201106	ABC	105	ABC	101
201106	ABC	107	DEF	201
201106	ABC	108	DEF	201
201106	ABC	221	DEF	201
201106	DEF	258	DEF	201
201206	ABC	108	ABC	107
201206	DEF	201	ABC	221

f) We calculated the average weighted GPA from students in ABC courses and ignored any other courses, which are not letter grading.
After doing the query, we found that the majors perform the best in ABC courses are O255, O151, O193, O167, O169, O275, O179, O176, O100, O113, O139, O207, and the average GPA of students in those majors is 4.00. The majors perform the worst in ABC courses are O152, O279, O263, O177, OT81, O261, O281, O285, and the average GPA of students in those majors is 0.

g)
For part 1 of this problem, we used *(the number of students ending in ABC who change their initial non-ABC majors)/(the total number of students ending in ABC)*. After doing query, we found that the percent of students transfer into one of the ABC majors is 20.816%.

For part 2 of this problem, we found all students who transfer into ABC majors, no matter if they are still in the major, and we used *(the number of transfer students from major X)/(total number of all transfer students to ABC)*. The results are shown below.

Major name	Percent
DEF2	18.0890126755%
DEF1	12.299209913%
OT16	12.4117546752%
OT 26	4.85192974478%
OT 35	5.69772068436%