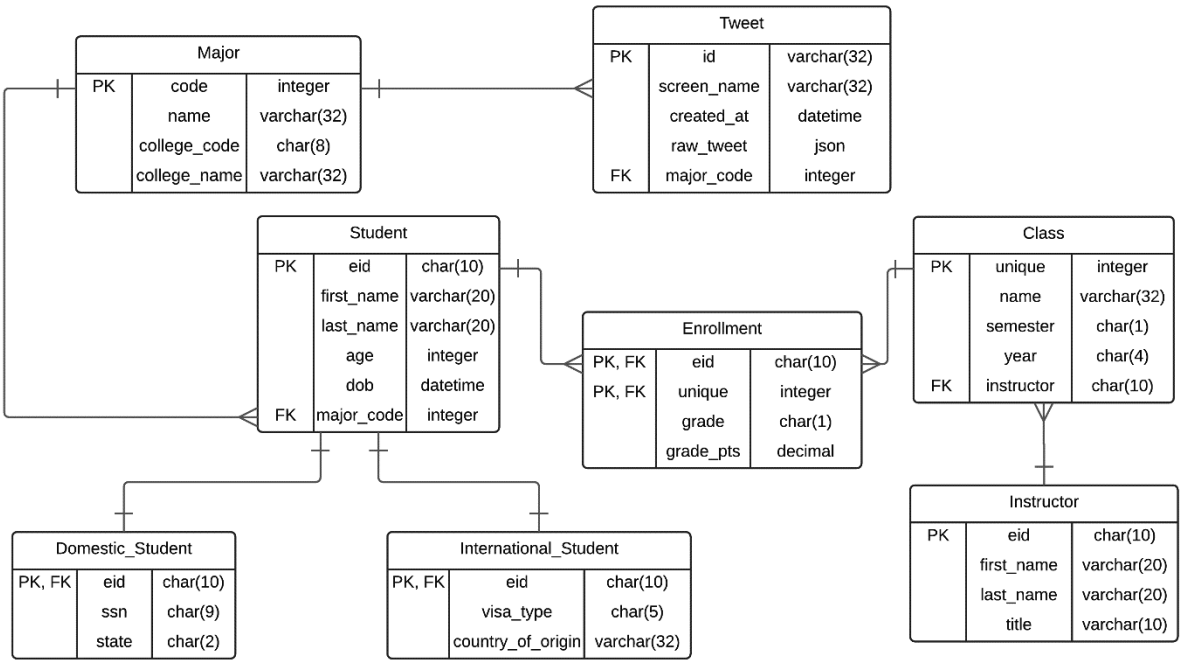
Practice Exam - CS 327E Fall 2016

This practice exam is based on latest rendition of the *utexas* database, a simple database of UT students, instructors, classes, enrollments, and most recently, majors and tweets.

Below is the logical ERD for this database:



Logical ERD - utexas Database - CS 327E Fall 2016

For convenience, each table in the database is also listed below. Note that the underlined fields are the primary keys for each table, while the **bold** fields are the foreign keys for each table.

Major(code, name, college_code, college_name)
Tweet(id, screen_name, created_at, raw_tweet, **major_code**)
Student(eid, first_name, last_name, age, dob, **major_code**)
Domestic_Student(**eid**, ssn, state)
International_Student(**eid**, visa_type, country_of_origin)
Instructor(eid, first_name, last_name, title)
Class(unique, name, semester, year, **instructor**)
Enrollment(**eid**, **unique**, grade, grade_pts)

Answer the questions about this database and on the following pages. You may remove this page from the practice exam for reference if that is convenient.

Part 1: Database Design:

Question 1. Is this schema in 1NF? Recall that to be in 1NF, every cell must have a scalar value. If the answer is no, identify which fields violate 1NF. Use the notation *table_name.column_name* to specify any violations found.

Tweet.raw_json (b/c raw_json is a json document, not a scalar value)

Question 2. Is this schema in 2NF? Recall that to be in 2NF, the schema must be in 1NF and every non-key field must be dependent on the **entire** key. Use the notation *table_name.column_name* to specify any violations found.

Tweet.raw_json (b/c raw_json is a json document, not a scalar value)

Question 3. Is this schema in 3NF? Recall that to be in 3NF, the schema must be in 2NF and every non-key field must be dependent **only** on the key. Use the notation *table_name.column_name* to specify any violations found. If you found any new violations, please provide a brief explanation.

Tweet.raw_json (b/c raw_json is a json document, not a scalar value)
Major.college_name (b/c Major.college_code -> Major.college_name and Major.college_code is not the pk for this table)

Question 4. One weakness of the current schema is that a student can only have one major. Suppose we wanted to represent double-majors, that is allowing a student to have up to two majors. Make the necessary changes to the schema to allow for double-majors. Specify the new table definitions below using the same table notation as above. Note: you only need to provide definitions for any new or changed tables.

Student(eid, first_name, last_name, age, dob)
Major(code, name, college_code, college_name)
StudentMajor(**eid**, **major code**)

Question 5. Suppose we wanted to represent the pre-requisites for a class. That is, a class can have zero or more pre-requisite classes. If a class has a pre-requisite, then it also has minimum grade to satisfy the pre-requisite. Make the changes to the schema to model pre-requisites classes and specify the new table definitions below. Again, please use the same table notation as above for your answer. Note: you only need to provide definitions for any new or changed tables.

Prereq(**unique**, **prereq_unique**, min_grade)

Part 2: SQL:

The following questions are asking you to translate the description to a valid SQL statement.

Question 1. Add a new Class to the database with the following details: unique = 98765, name = Data Science, semester = S, year = 2017, instructor = jeffwu. Assume that this instructor already exists in the Instructor table.

```
insert into Class(unique, name, semester, year, instructor) values(98765, 'Data Science', 'S', '2017',
'jeffwu');
```

Question 2. Change the Data Science class to name = Data Science and Prediction and semester to null.

```
update Class set name = 'Data Science and Prediction', semester = null where unique = 98765;
```

Question 3. Delete the Data Science class from the database.

```
delete from Enrollment where unique = 98765;
delete from Class where unique = 98765;
```

Question 4. Delete all tweets for major_code = CS and created_at = 12/01/2016.

```
delete from Tweets where major_code = 'CS' and created_at = '2016-12-01';
```

Question 5. List the distinct class names in the database. Sort the results by class name.

```
select distinct name from Class order by name;
```

Question 6. List the eids and names of all domestic students who are from out-of-state. Sort the results by eid.

```
select eid, name
from Student s join Domestic_Student d
on s.eid = d.eid
where d.state <> 'TX'
order by eid;
```

Question 7. List the eids and names of all the Math majors who are currently enrolled in the class = Elements of Databases. Assume that "currently enrolled" means a null grade in the Enrollment table and Math majors have the major_code 'math'. Sort the results by eid.

```
select eid, name
from Student s join Enrollment e
on s.eid = e.eid
join Class c
on e.unique = c.unique
where c.name = 'Elements of Databases'
and e.grade is null
and s.major_code = 'math'
order by eid;
```

Question 8. Based on your answer in Part 1, find the most common pre-requisite classes in the database. Your answer should return the unique of the pre-requisite class as well as the number of classes that use it as a pre-requisite. Return the top 10 results.

```
select prereq_unique, count(*) as count
from Prereq
group by prereq_unique
order by count desc
limit 10;
```

Question 9. List the eids and names of all instructors who have **not** taught a class. The results should be sorted by the eid of the instructor.

```
select i.eid, i.first_name, i.last_name
from Instructor I left outer join Class c
on i.eid = c.instructor
where c.instructor is null
order by i.eid;
```

Question 10. Find the instructors who have taught more than 3 classes in 2016. Return the eids and number of classes taught. Sort the results by the number of classes in descending order.

```
select instructor, count(*) as count
from Class
where year = '2016'
group by instructor
having count(*) > 3
order by count desc;
```

Question 11. Create a view that shows the number of enrollments for each student in 2016. The view should return the eid and enrollment count.

```
create view Student_Enroll_2016 as
select s.eid, count(e.unique) as enroll_count
from Student s left outer join Enrollment e
on s.eid = e.eid
left outer join Class c
on e.unique = c.unique
where c.year = '2016'
group by s.eid;
```

Question 12. Using the view from the previous question, return the minimum, maximum, and average number of enrollments for 2016. The query should not count those students who have zero enrollments this year.

```
select min(enroll_count), max(enroll_count), avg(enroll_count)
from Student_Enroll_2016
where enroll_count > 0;
```