## **COSC 30603**

**Lab Assignment 4: EER Model and Database Constraints**

You need to turn in: a hardcopy of your answers (type your answer in a different font or color, e.g. **bold face style**), a hardcopy of lab4.sql and an e-copy of your lab4.sql.

You need to work in MySQL workbench or other SQL client software, e.g. Navicat (not free), SQLYog Community Version, Sequel Pro (for Mac).

Don’t use terminal.

**Name: Hieu Nguyen**

Due: Oct 9, 2019.

1. The concept of Instructor was not appropriately captured in the EER diagram of Lab Assignment 3 (see next next page). The new requirements are as follows.

(a) Those graduate students who really teach courses should not be considered as TA. They should be included in the Instructor category.

(b) University also allows people from local industries to teach courses.

Due to the new requirements, Instructor becomes an important data on its own. We decide to make Instructor as an entity type. Answer the following questions related to incorporating these new requirements into the EER diagram.

• Can Graduate (student) be a subclass of Instructor?

|  |
| --- |
| No |

• Can Graduate be a superclass of Instructor?

|  |
| --- |
| Yes |

• Can we make a separate entity type GraduateInstructor as a subclass of both Graduate and Instructor?

|  |
| --- |
| No |

• Can NonUniversityInstructor be a subclass of Instructor?

|  |
| --- |
| No |

• Can Faculty be a superclass of Instructor?

|  |
| --- |
| Yes |

• Can Faculty be a subclass of Instructor?

|  |
| --- |
| No |

• Can we make a separate entity type FacultyInstructor as a subclass of both Faculty and Instructor?

|  |
| --- |
| No |

1. Make appropriate changes to the EER diagram to incorporate the above changes in requirements. Draw the corresponding part of the EER diagram on a separate sheet of paper and submit the sheet along with this lab assignment.

Graduate

Faculty

advisor

U

Instructor

1. Write the relational schema for the above changes in a file called lab4\_eer.sql. Your schema only needs to have the CREATE TABLE statements for those superclasses and subclasses that are involved in the changes, such as Faculty, Instructor and Graduate. You need to submit a printed copy of the file.

Office hour

M

P

N

use

Lab

Project

N

1

director

N

1

supervise

N

1

N

1

teach

1

N

1

N

1

counselor

advisor

1

N

instructor

M

N

has

Textbook

N

1

has

N

1

works for

1

1

chair

M

N

prerequisite

M

N

offers

1

1

coordinator

Faculty

Dept requirement

N

offers

university

college

requirements

name

type

M

N

TA

Course Description

N

M

grade

taken

Course Offering

Graduate

Probational Student

Undergraduate

M

1

N

N

minor

major

Degree Program

Department

Student

gpa

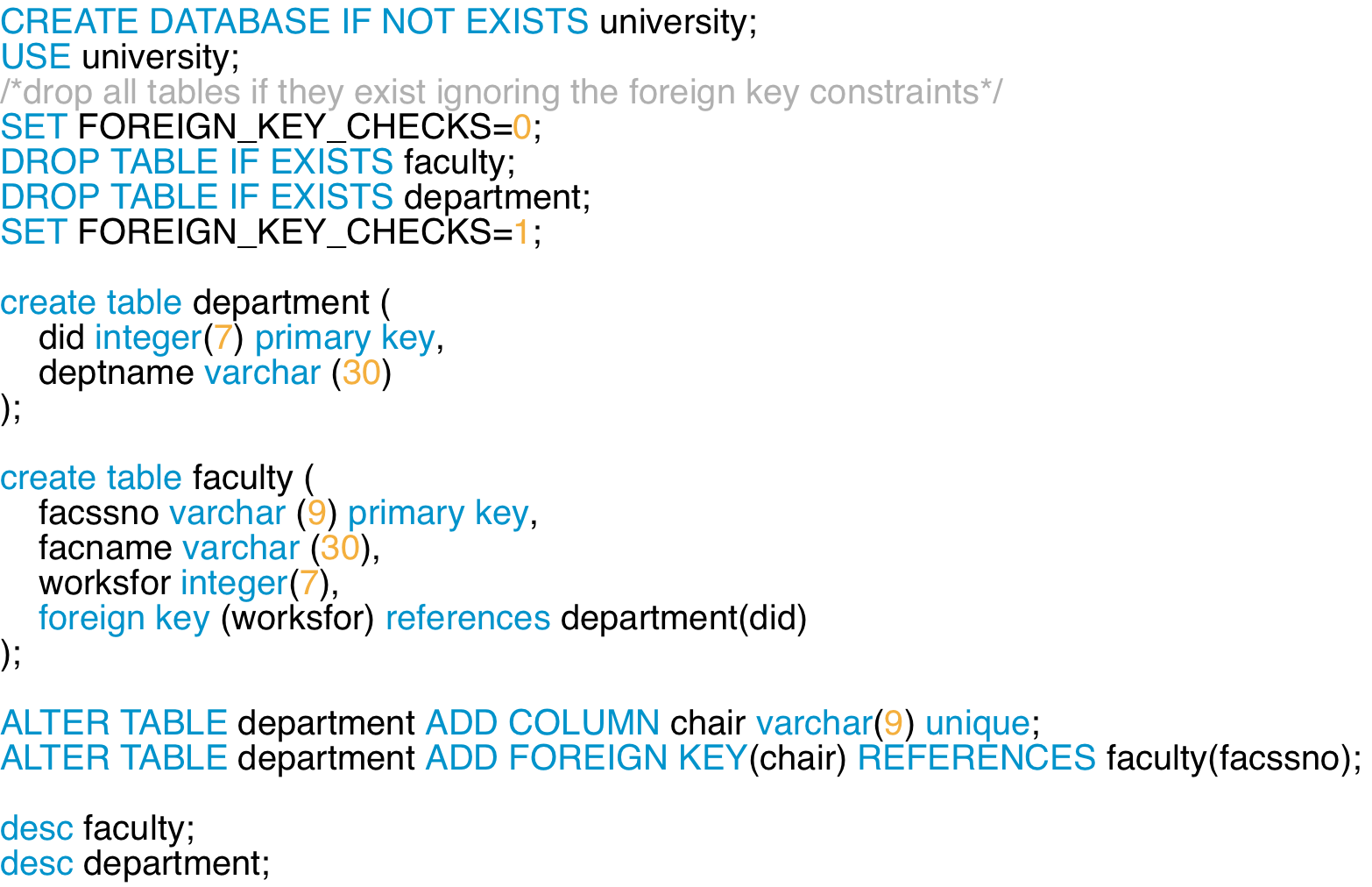
mentor

Office hour

1. When referential relationships are involved between two tables, the order of creating and deleting the two tables needs to be taken care of. We already know that, if table A references table B, then table B should be created before table A and table A should be dropped before table B. What if tables A and B reference each other through different relationship types?

From the EER Diagram, we can see that the relationships *chair* and *works for* let Faculty and Department reference each other. Apparently we cannot create the tables with their foreign keys directly. A solution to the problem is given below.

Create a file called lab4.sql and add the following commands



The statement “drop table ...” allows us to drop the table along with all the referential constraints that reference the table. Therefore, although Faculty and Department are referencing each other, they can still be dropped.

When we first create the table Department, we do not include the foreign key chair. After Faculty table is created, we use the “alter table” statement to add the foreign key in Department.

1. When we insert a tuple that referencing a non-existing entity, it violates the foreign key constraint. We have to insert the referenced (parent) tuple first. When mutual references are involved, the insertion of the first tuple is a problem. The following example shows how to solve this problem.

Append the following command to lab4.sql.

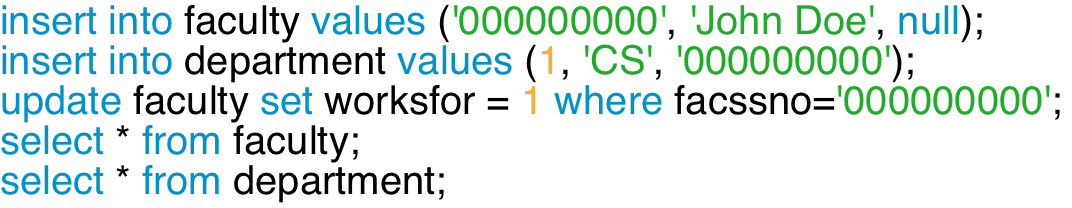


Run lab4.sql in SQL\*Plus. Why do you see the problem?

Answer:

* The problem occurs because an error 1452, which related to a foreign key constraint fail. It said we cannot add or update a child row.

Now remove the above statement from lab4.sql and append the following commands.



Execute again. Explain why there is no problem now.

Answer:

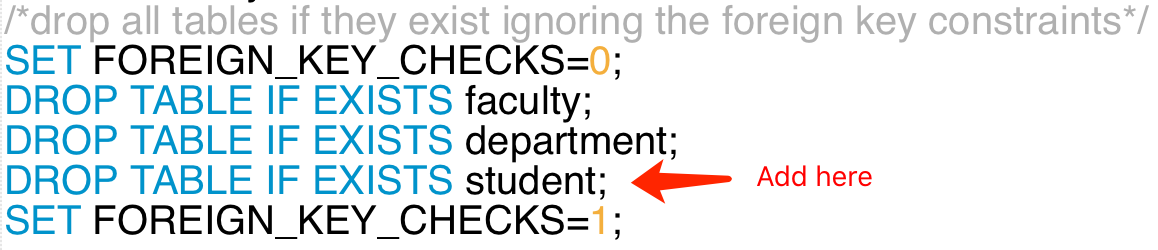
* There isn’t any error as we already add the column chair to table “department” then we refer it to ‘faculty’ table. The attributes we add on is valid since there is already a tuple in ‘faculty’ table. It works same as we update worksfor.

1. Add the following commands at the end of lab4.sql.

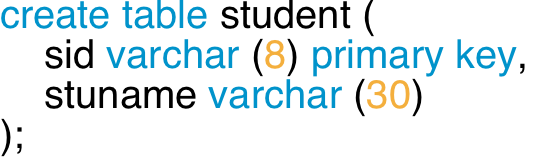
First add the drop statement in the beginning drop statements.



So it looks like:



Then add the following commands at the end of lab4.sql



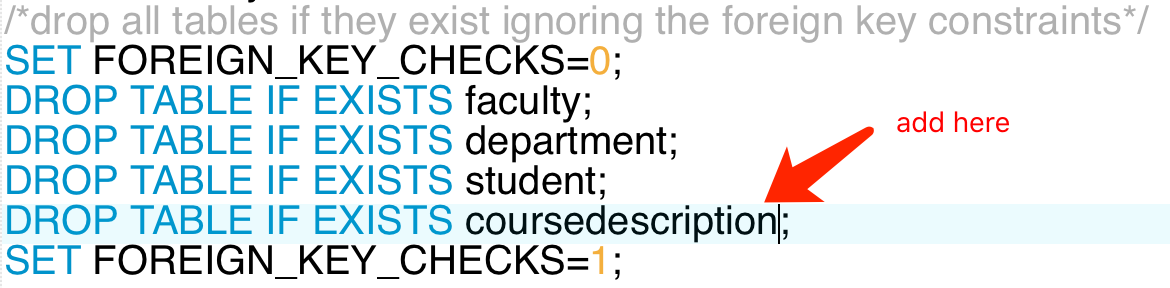
Now implement in lab4.sql the recursive 1:N relationship type *mentor* the same way as step 4 (a table referencing itself cannot be created directly). Remember that a student can have at most one mentor and can be a mentor of several students.

Copy your answer here:

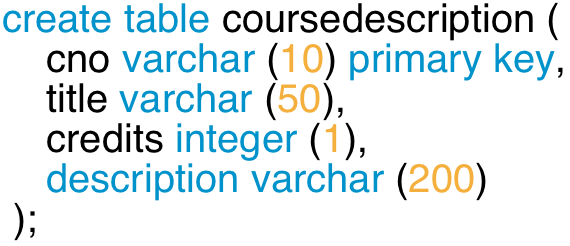
* Alter table student add column mentored varchar(8) unique;
* Alter table student add foreign key(mentored) references student(sid);

For N:M recursive relationship types, there is no such ordering problem. You always need to create a new table for an N:M relationship after creating the referenced tables.

First add the drop statement in the beginning drop statements.



Add the following commands at the end of lab4.sql.



Now implement in lab4.sql the recursive N:M relationship type *prerequisite*.

Copy your answer here:

* Create table prerequisite(

coursed varchar(10),

requiredid varchar(10),

primary key (coursed, requiredid),

foreign key (courseid) references coursedescription(cno),

foreign key (requiredid) references coursedescription(cno));

1. We have seen referential integrity violations due to insertions. We will now discuss referential integrity violations due to deletions.

(a) Enter the following commands:



Why does it not work?

Answer:

* Since John Doe is already in ‘faculty’ table with his did = 1, so we can delete a key that had been referenced

Try the following statement:

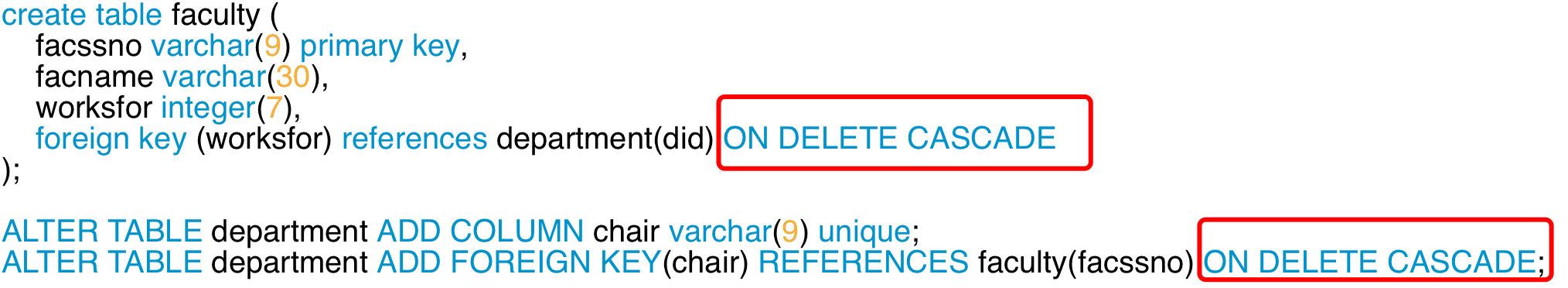


Does this command work? Why?

Answer:

* It does not work since CS is already an attribute in ‘department’ table with its value as facssno = ‘000000000’

(b) Now we will try the delete cascade option of foreign key constraints. In lab4.sql, change the alter table command for table Department as follows:



Run lab4.sql. Now enter the following commands again.

select \* from faculty;

delete from faculty where facssno = '000000000';

Does it work now?

Answer: Yes, it does

Type:

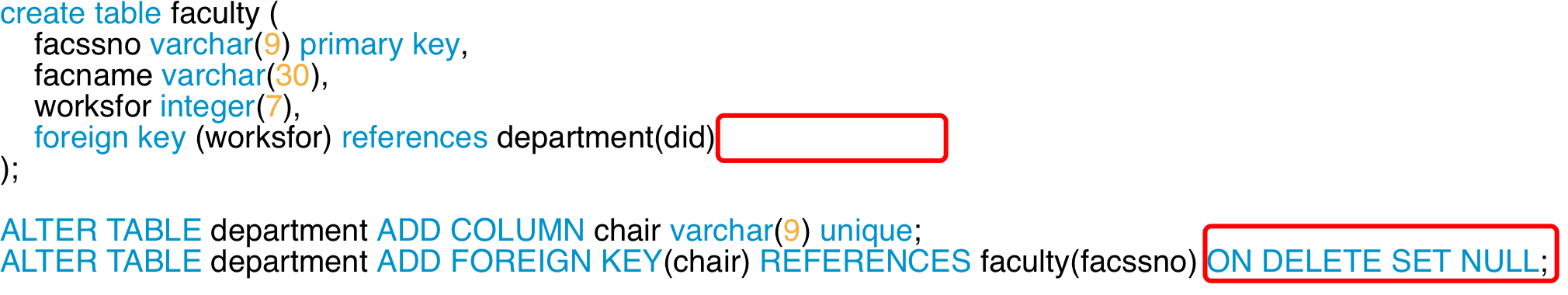
select \* from department;

Why is there no record in table department now?

Answer:

* Because the value refers to facssno = ‘000000000’ is also deleted

(c) Now we will try the SET NULL option of foreign key constraints. In lab4.sql, change the alter table command for table Department as follows:



Run lab4.sql. Now enter the following commands again.

select \* from faculty;

delete from faculty where facssno = '000000000';

Does it work?

Answer: Yes, it does

Type:

select \* from department;

What happed to the CS department tuple?

Answer:

* Since we deleted the tuple in ‘faculty’ table, the attribute for CS department is set to be null

(d) We have three levels of actions for deletions involving foreign key constraints.

• By default, deletion is rejected, if a tuple being deleted is being referenced by some other tuples.

• SET NULL: Deletion is allowed. When a referenced tuple is deleted, the foreign key attributes of the referencing tuples are set to null.

• DELETE CASCADE: Deletion is allowed. When a referenced tuple is deleted, the tuples that reference the deleted tuple are automatically deleted. This option may be dangerous since it can incur automatic deletion of records.

Based on the discussion, which option will you choose for relationship *chair*? How about relationship *works for*? Explain your choices.

Answer:

* For the chair relationship, we use the SET NULL because it prevent the data lost in which data are linked to each other by foreign key. Furthermore, by setting the value to null, we are able to update it whenever we want. Similarly, with the relation worksfor, we use the SET NULL.

Print out the file lab4.sql and submit it along with this lab assignment. ALSO SUBMIT E-COPY of lab4.sql.