Name: Zijie Yu

Database Management Systems (CSC434)

Final Project

README: I made college database (including foreign key, primary key and data) SQL file called college.sql in the file directory. You can always run or check this file to see my database.

1) - Define the information content of your database.

a)-Define a set of entities and appropriate attributes for each entity. Minimum 10 entities.

Note: The first attribute is primary key, the Blue color entity means entity created automatically by Visual Paradigm because of many to many relationship (intermediate table).

- **Students**: Student_ID; First_Name; Last_Name; Country; Age; Phone; Email; Expected_graduation_year; Address; Hobby;
- **Major**: Major_Name; Department; Minimum_GPA; DepartmentId
- **Students_Major**: Students_StudentId; Major_MajorId
- **Course**: CourseId; Course_Name; Seats Avaliable; Professor_ProfessorId; SubjectId; ClassSceduleId;
- **Student_Course**: Students_StudentId; Course_CourseId

- **Classroom**: ClassroomId; Classroom_Name; Buildings; Location; Capacity;
- **Classroom_Course:** Classroom_ClassroomId; Course_CourseId
- **Course_Scedule**: CourseSceduleId; Meeting_Day; Time_Start; Time_End
- **Subject**: SubjectId; Subject_Name; DepartmentId
- Professors: ProfessorId; First_Name; Last_Name; Email; Phone; Address
- **Subject_Professors:** Subject_SubjectId; Professors_ProfessorId
- **Titles**: TitleId; Type
- **Titles:** Titles_Titled; Professors_ProfessorId
- **Manager**: ManagerId; First_Name; Last_Name; Phone; Address
- **Advisor**: AdvisorId; First_Name; Last_Name; Phone; Address
- **Department**: DepartmentId; Name; AdvisorId; ManagerId

b)-Define a set of relationships that might exist between/among entities and attributes. Such relationships may include one-to-one, one-to-many and many-to-many associations.

Students – Course: Many to many

(One student can take many course, and one course can have many students)

Students – Major: Many to many

(One student can have one or more major, and one major can have many students.)

Major – Department: Many to one

(One major must belong to one department, but one department can have many major)

Department – Manager: One to one

(One department can only have one manager, and one manager can only manage one specific department)

Department – Advisor: One to one

(One department can only have one advisor, and one advisor can only manage one specific department)

Department – Subject: One to many

(One department can have many subjects, but one subject can only belong to one department.)

Subject – Course: One to many

(One subject can have many different courses, but one course must have one subject)

Course – Course_Schedule: Many to one

(One course_Schedule can have many different courses, but one course can only have on specific schedule.)

Course – Classroom: Many to many

(One course may have one or more classroom, and one classroom can have many different courses.)

Professor – titles: Many to many

(One professor can have one or more titles, and one titles can have many different professors.)

Professor – subject: Many to many

(One professor can teach many different subject course, and one subject course can have many different professors.)

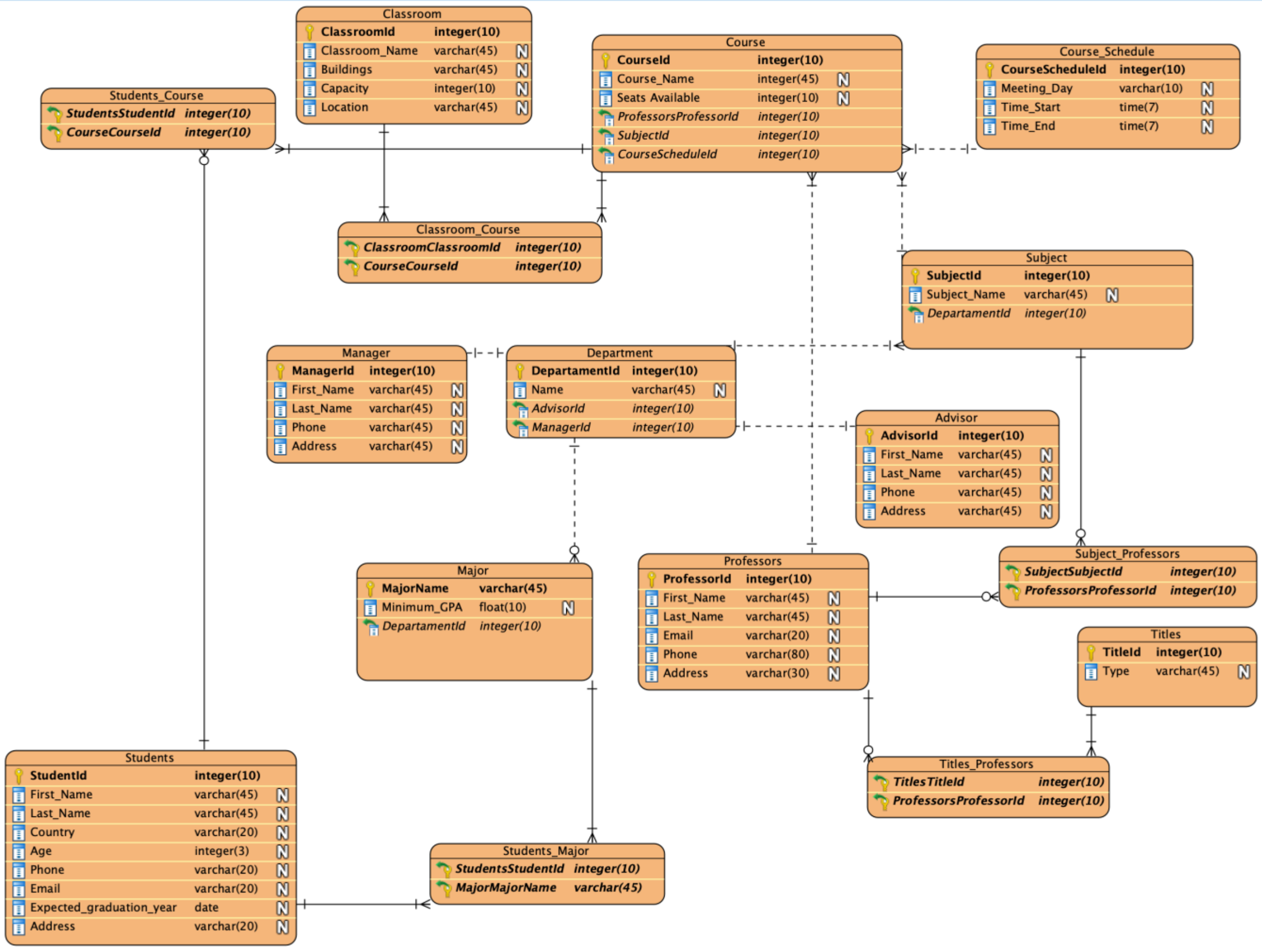
c)-Define a set of constraints that may be imposed on data.

- One department can only have 1 manager.
- One department can only have 1 advisor.
- Student must take at least one course
- Every Professor must have proper title, such as Ph.d, Academic or assistant

- One major can only belong to one department.
- One course can only belong to one subject, such as CSC or MATH.
- A course must have at least one classroom.
- A course must have a course schedule
- One manager can only manage one department.
- One advisor can only work for one department.

(For example, CAS advisor are not allowed to help ART department students.)

2) - Define an E-R Diagram for your database design.



3) - Define a relational schema for your database design.

a)-Define one or more realistic key(s) for every relation scheme. Use both simple and composite keys.

- **Student:** The primary key should be SutdentId, since StudentId is unique.
- **Students_Course:** Both Students_StudentId and Course_CourseId are foreign keys.

- **Course**: CourseId should be primary key, because it is unique. And Professors_professorId, SubjectId, CourseScheduleId are foregin keys.
- **Students_Major**: Both Students_StudentId and Major_MajorName are foreign keys.
- **Major**: MajorName is primary key because it is unique. The DepartmentId is foregn key.
- **Department**: The primary key should be DepartmentId, the foregn keys are AdvisorId and ManageId.
- **Manager**: ManagerId should be primary key since it is unique.
- **Advisor**: AdvisorId should be primary key.
- **Course_Schedule**: The primary key should be CourseScheduledId since it is unique.
- **Subject**: SubjectId should primary key and the DepartmentId should be foreign key.
- **Ttitles**: The titleId should be primary key.
- **Titles_Professor**: Both Titles_TtitleId and Professor_ProfessorId are foreign keys.

- **Professors:** ProfessorId should be primary key since it is unique.
- **Subject_Professors:** Both Subject_SubjectId and Professors_ProfessorId are foreign keys.
- **Classroom:** The primary key should be classroomId since it is unique.
- **Classroom_Course:** Both Classroom_ClassroomId and Course_CourseId are foreign keys.

b)-Define a realistic set of Functional / Multi-Valued Dependencies (when appropriate) for every relation scheme.

Note: "→" means functional dependencies and "→→" means multi-valued dependencies

Student:

StudentId → First_Name

StudentId → Last_Name

StudentId → Country

StudentId → Age

StudentId → Phone

StudentId → Email

StudentId →→ Address (Student may have several address)

StudentId →→ Hobby (Student may have several hobby)

Major:

MajorName → Department

MajorName → Minimum_GPA

Manager:

ManagerId → First_Name

ManagerId → Last_Name

ManagerId → Phone

ManagerId →→ Address (Manager may have several address)

Classroom:

ClassroomId → Classroom_Name

ClassroomId → Buildings

ClassroomId → Capacity

ClassroomId → Location

Course:

CourseId → Course_Name

CourseId → Seats Avaliable

Department:

DepartmentId → Name

Professors:

ProfessorId → First_Name

ProfessorId → Last_Name

ProfessorId → Email

ProfessorId → Phone

ProfessorId →→ Address (Professor may have several adress)

Advisor:

AdvisorId → First_Name

AdvisorId → Last_Name

AdvisorId → Phone

AdvisorId →→ Address (Advisor may have several address)

Titles

TitleId → Type

Subjec

SubjectId → Subject_Name

Course_Schedule

CourseScheduledId → Meeting_Day

CourseScheduledId → Time_Start

CourseScheduledId → Time_End

C-Check whether your relational schema is in 2NF, 3NF, BCNF, 4NF.

2NF:

Since 2NF is every non-key field must depend on the entire primary key, not on part of composite primary key.

My “college database” only has one single primary key (for all entities), so it is automatically in 2NF.

3NF:

Since 3NF is a non-key field cannot depend on another non-key field. (or do not have transitive dependency.)

I can’t find any transitive dependency on my college database, so my database is in 3NF.

BCNF:

Since BCNF means that for any dependency A→B, A should be a super key.

In my classroom entity,

Classroom: ClassroomId; Classroom_Name; Buildings; Location; Capacity;

Sometimes, classroom_name is unique, so it is determinant the classroomId.

How to solve this?

I'm going to delete classroom_Name or delete ClassroomId. After doing this, all my database is in BCNF.

4NF: Since 4NF means that it should be not have multi-valued dependency in database.

As I checked my database, I have several multi-valued lists in previous question. So, my database is not in 4NF.

To be in 4NF, I just need to decompose the table into 2 tables.

Since we consider a relation in BCNF to be fully normalized, I don't have to change my database to archive in 4NF.

d)-Put your relational schema in the highest normal form that is possible.

As I mentioned in the previous question, my database is in 3NF. Since in many cases, 3NF can be seen as highest normal form, I don’t need to do any change.

If I want my database in 4NF, then I will change my database below:

Constrain that classroom is unique and delete classroom_ID to meet BCNF.

To be in 4NF, I need to delete multi_valued function by using decompose the table into 2 tables.

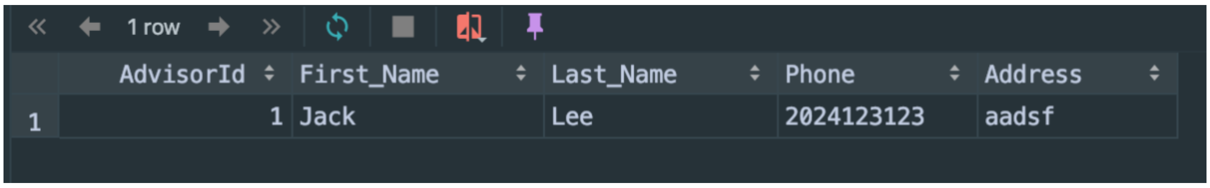
4) Implementation: Create your database using Oracle, or MySQL, or… to Perform the following operations.

A) You are required to execute SQL queries that include the following operations. For each query, provide the SQL statements along with the output. For each of the following, try different SQL statements (i.e., using one relation, more than one relations,...).

select involving one/more conditions in Where Clause

```
select * from advisor
where Last_Name = "Lee";
```

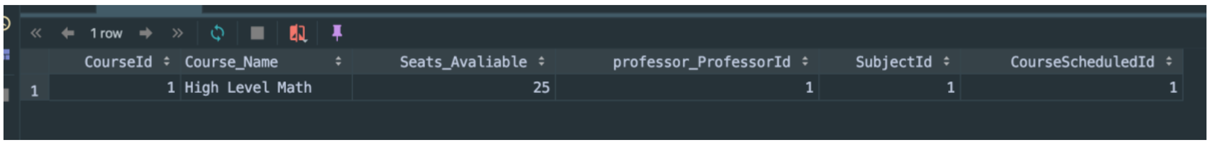
Output:



```
1 row
   AdvisorId  First_Name  Last_Name  Phone       Address
1          1  Jack        Lee        2024123123  aadsf
```

```
select * from Course
where SubjectId = 1
and CourseScheduledId = 1;
```

Output:



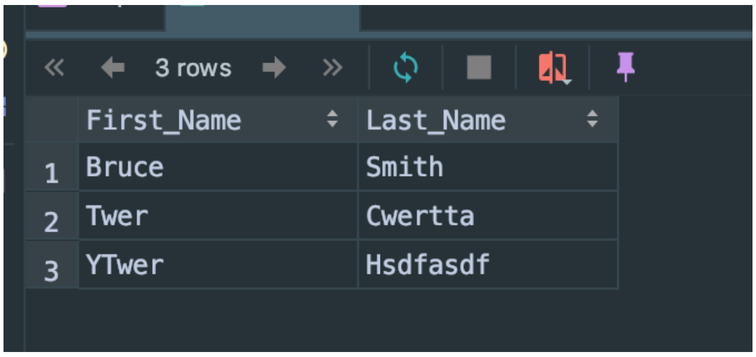
```
1 row
   CourseId  Course_Name      Seats_Avaliable  professor_ProfessorId  SubjectId  CourseScheduledId
1         1  High Level Math               25                      1          1                  1
```

For another more difficult example, if I want to find students whose major is computer science:

```
select First_Name, Last_Name from Students, Students_Major, Major
where studentsId = Students_StudentId
```

```
and MajorName = Major_MajorName
and majorName = "computer science";
```

output:



```
3 rows
    First_Name   Last_Name
1   Bruce        Smith
2   Twer         Cwertta
3   YTwer        Hsdfasdf
```

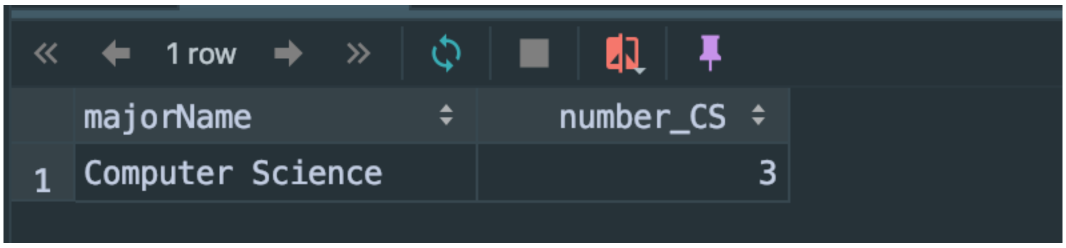
select with aggregate functions (i.e., SUM,MIN,MAX,AVG,COUNT)

Count:

If I want to know how many students' major is computer science. Then the SQL should be:

```
select majorName, count(StudentsId) as number_CS from students,
Students_Major, Major
where StudentsId = Students_StudentId
and Major_MajorName = MajorName
and MajorName = "Computer Science"
group by majorName;
```

Output:



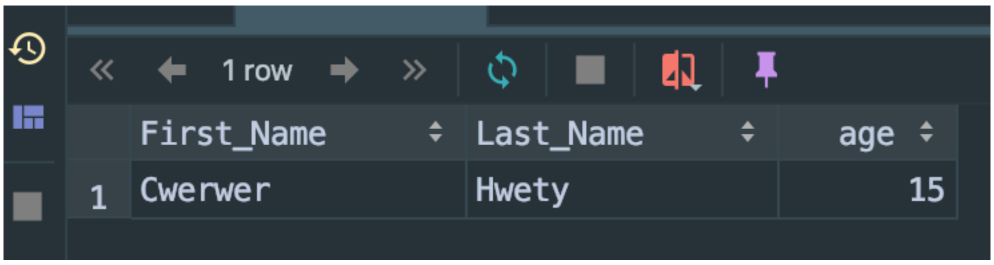
```
1 row
    majorName          number_CS
1   Computer Science           3
```

Min:

If I want to find youngest students in college, I will use SQL below:

```
select First_Name, Last_Name, age from students where
age = (select min(age) from Students);
```

Output:



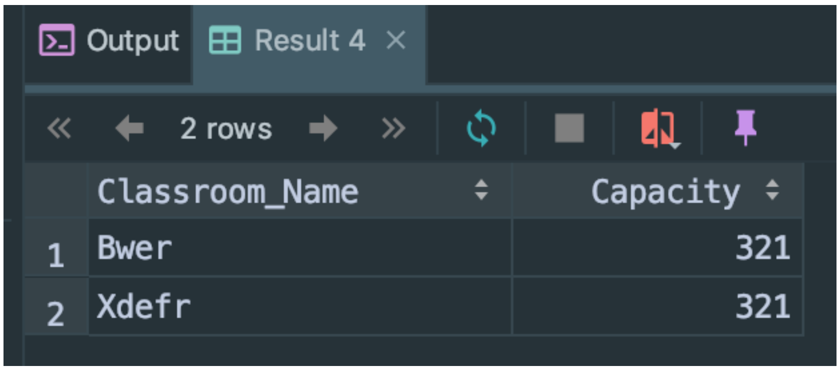
```
   First_Name   Last_Name   age
1  Cwerwer      Hwety        15
```

Max:

If I want to find which classroom can contain more students than any other classroom, I will use sql below:

```
select Classroom_Name, Capacity from Classroom
where Capacity = (select max(Capacity) from Classroom)
```

Output:


Output | Result 4

2 rows

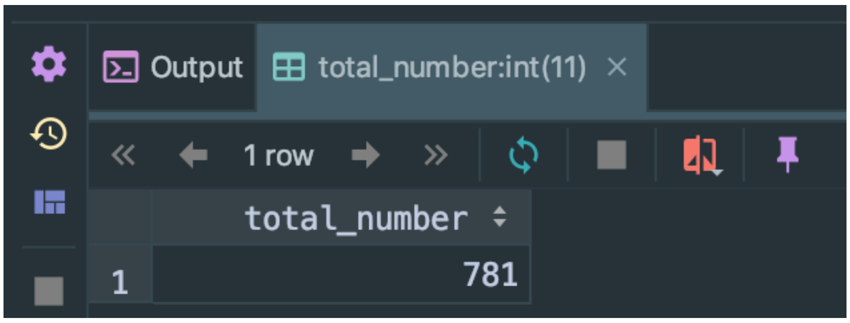
	Classroom_Name	Capacity
1	Bwer	321
2	Xdefr	321

Sum:

If I want to find how many total classroom capacity available, I will use SQL below:

```
select sum(Capacity) as total_number from Classroom;
```

Output:


Output | total_number:int(11)

1 row

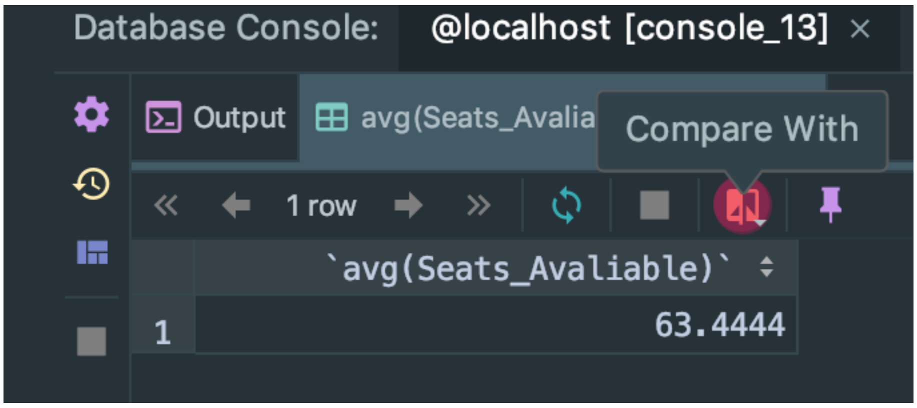
	total_number
1	781

AVG:

If I want to find the average seats available for each course, I will use SQL below:

```
select avg(Seats_Avaliable) from Course;
```

Output:



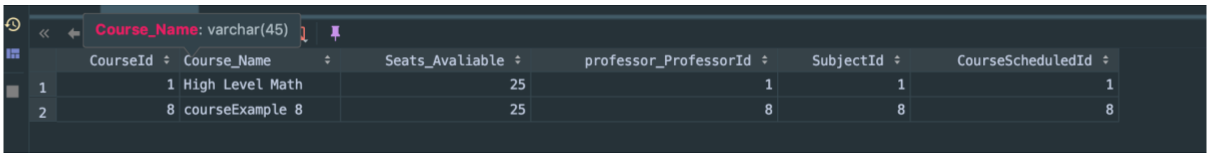
	`avg(Seats_Avaliable)`
1	63.4444

select with Having, Group By, Order By clause

Only "Having":

```
select * from Course
having Seats_Avaliable = 25;
```

Output (In this case, having = where):

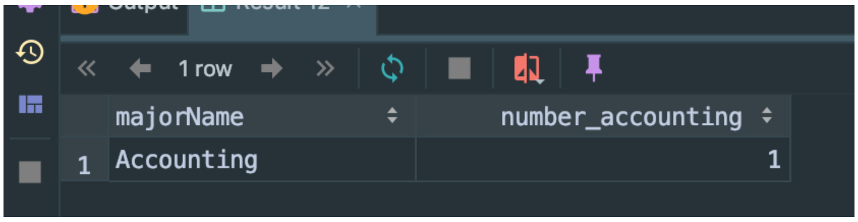


	CourseId	Course_Name	Seats_Avaliable	professor_ProfessorId	SubjectId	CourseScheduledId
1	1	High Level Math	25	1	1	1
2	8	courseExample 8	25	8	8	8

Having and group by:

```
select majorName, count(StudentsId) as number_accounting from
students, Students_Major, Major
where StudentsId = Students_StudentId
and Major_MajorName = MajorName
group by majorName
having MajorName =  "Accounting";
```

Output



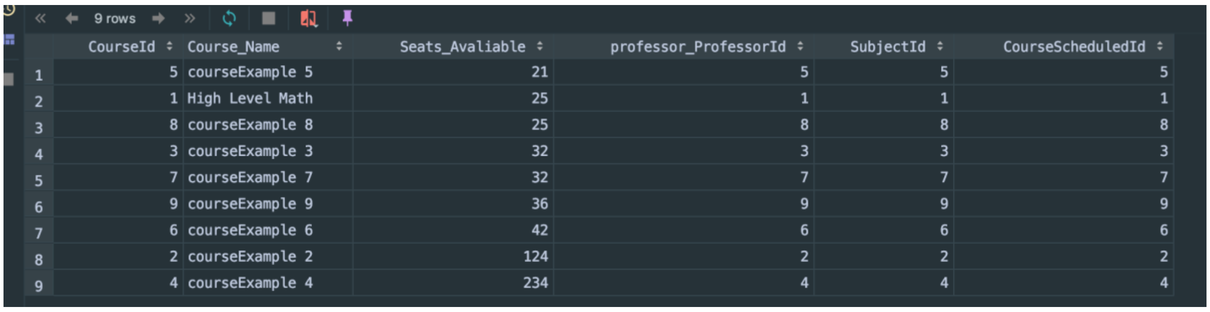
	majorName	number_accounting
1	Accounting	1

Order by:

If I want to compare course's seats available, I will write SQL below:

```
select * from Course
order by Seats_Avaliable;
```

Output:



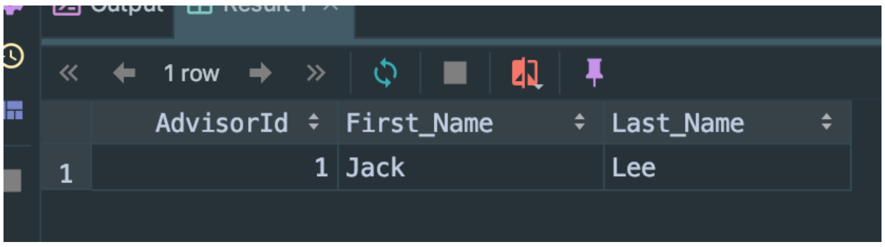
	CourseId	Course_Name	Seats_Avaliable	professor_ProfessorId	SubjectId	CourseScheduledId
1	5	courseExample 5	21	5	5	5
2	1	High Level Math	25	1	1	1
3	8	courseExample 8	25	8	8	8
4	3	courseExample 3	32	3	3	3
5	7	courseExample 7	32	7	7	7
6	9	courseExample 9	36	9	9	9
7	6	courseExample 6	42	6	6	6
8	2	courseExample 2	124	2	2	2
9	4	courseExample 4	234	4	4	4

Nested Select:

If I want to know who CAS advisor is, I will do SQL below:

```
select AdvisorId, First_Name, Last_Name from Advisor
where AdvisorId = (select AdvisorId from Department
    where name = "CAS")
```

output:



1 row

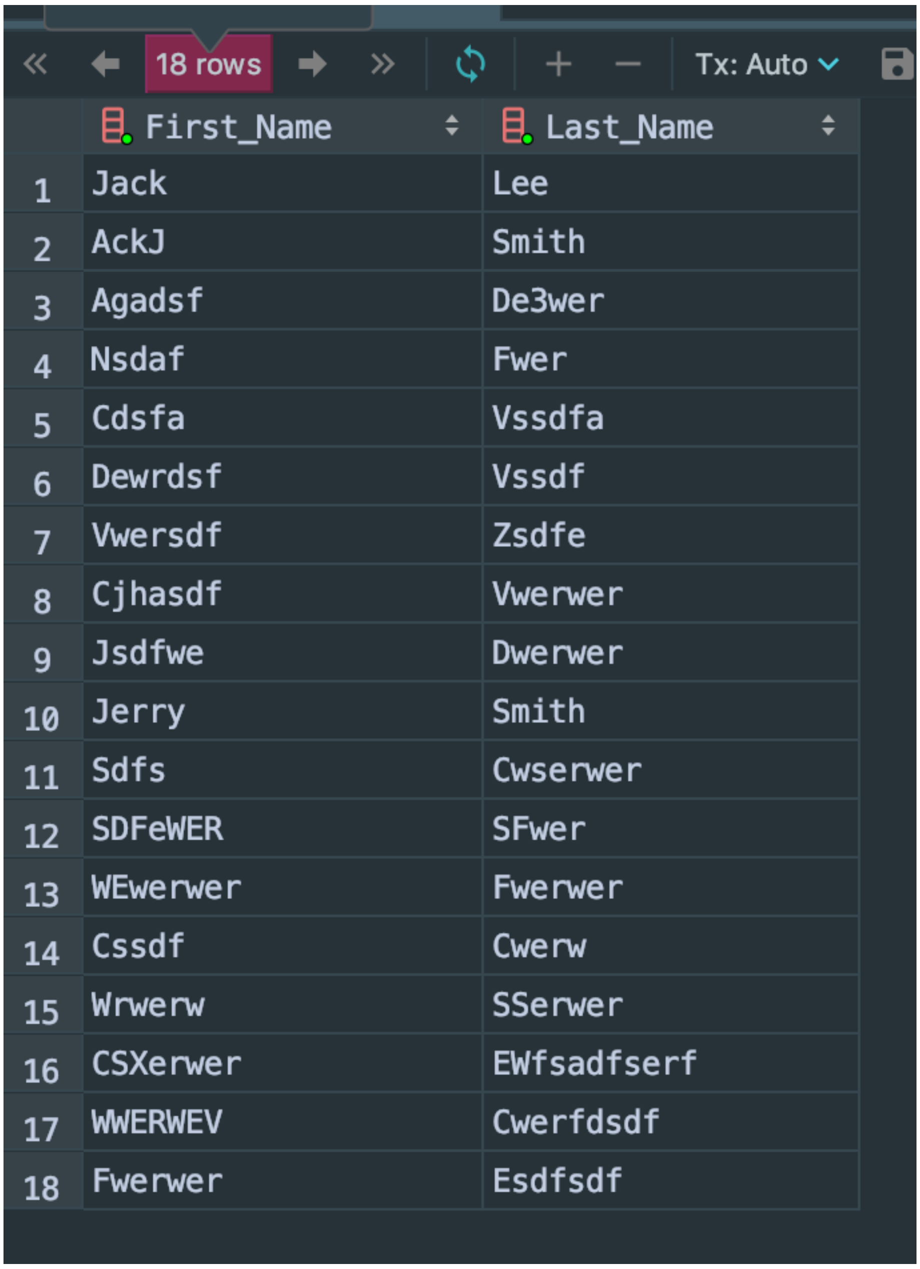
	AdvisorId	First_Name	Last_Name
1	1	Jack	Lee

select involving the Union operation

If I want to see all advisor and all manager in all departments, I will use sql below:

```
select First_Name, Last_Name from Advisor
union
select First_Name, Last_Name from Manager
```

output:

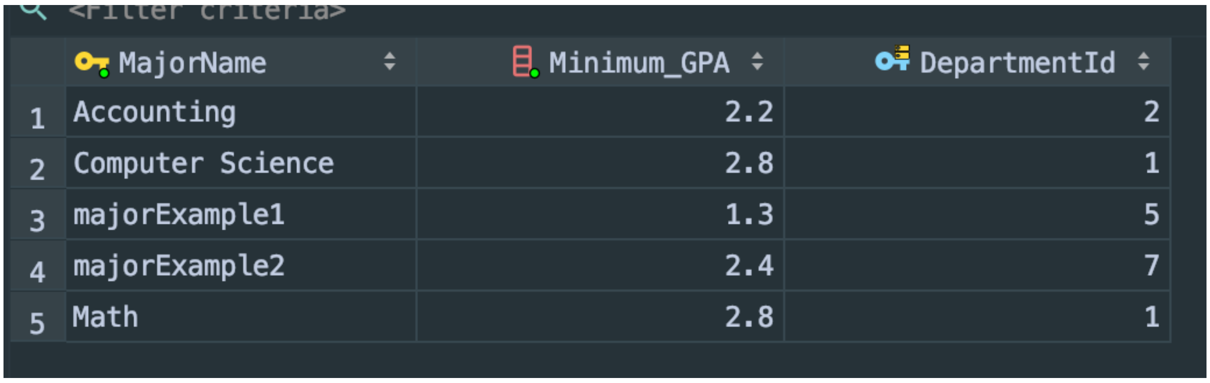


	First_Name	Last_Name
1	Jack	Lee
2	AckJ	Smith
3	Agadsf	De3wer
4	Nsdaf	Fwer
5	Cdsfa	Vssdfa
6	Dewrdsf	Vssdf
7	Vwersdf	Zsdfe
8	Cjhasdf	Vwerwer
9	Jsdfwe	Dwerwer
10	Jerry	Smith
11	Sdfs	Cwserwer
12	SDFeWER	SFwer
13	WEwerwer	Fwerwer
14	Cssdf	Cwerw
15	Wrwerw	SSerwer
16	CSXerwer	EWfsadfserf
17	WWERWEV	Cwerfdsdf
18	Fwerwer	Esdfsdf

- Insert:

insert one tuple into a table:

Table before insert:



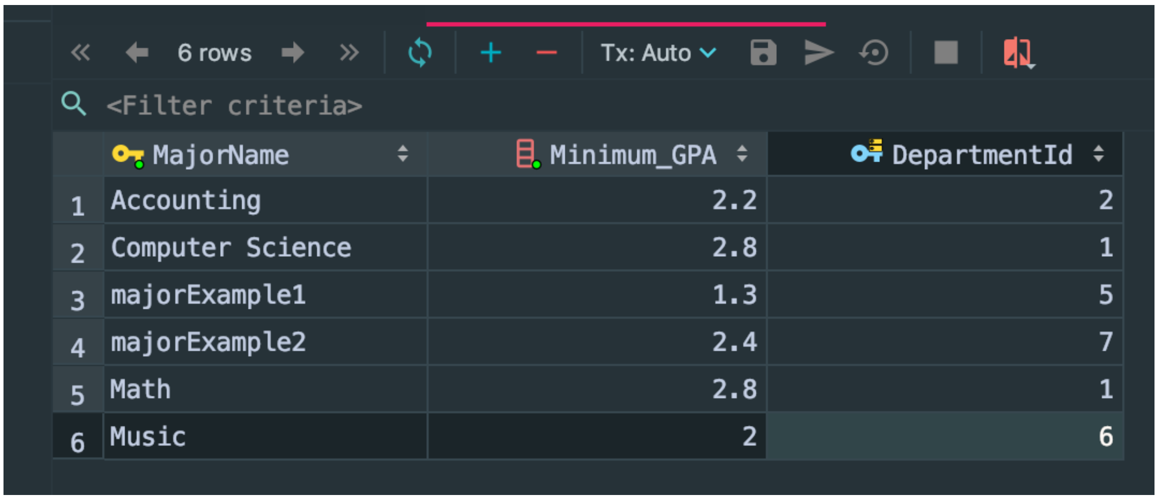
<Filter criteria>

	MajorName	Minimum_GPA	DepartmentId
1	Accounting	2.2	2
2	Computer Science	2.8	1
3	majorExample1	1.3	5
4	majorExample2	2.4	7
5	Math	2.8	1

SQL:

```
insert into Major (MajorName, Minimum_GPA, DepartmentId)
values ("Music", 2.0, 6)
```

Table after insert:



<Filter criteria>

	MajorName	Minimum_GPA	DepartmentId
1	Accounting	2.2	2
2	Computer Science	2.8	1
3	majorExample1	1.3	5
4	majorExample2	2.4	7
5	Math	2.8	1
6	Music	2	6

insert a set of tuples (by using another select statement)

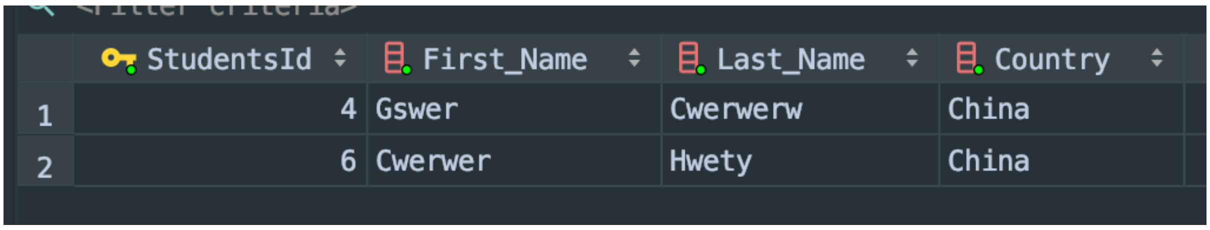
First, let’s create table called ChineseStudents by using SQL below:

```
create table ChineseStudents
(
   StudentsId              int auto_increment
      primary key,
   First_Name              varchar(45) not null,
   Last_Name               varchar(45) not null,
   Country                 varchar(20) not null,
   Age                     int         not null,
   Phone                   varchar(20) not null,
   Email                   varchar(20) not null,
   Expacted_graduation_year date       not null,
   Address                 varchar(20) not null
);
```

Then, insert all Chinese student to the table:

```
insert into ChineseStudents
select *  from Students
where country = "China";
```

The Chinese table become to:

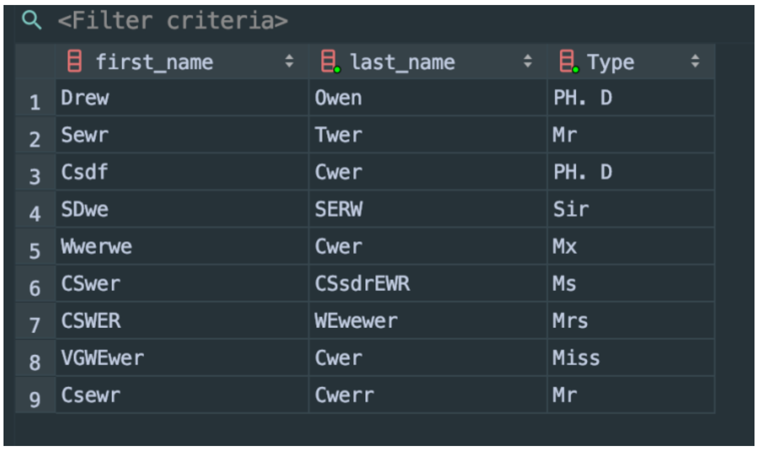


	StudentsId	First_Name	Last_Name	Country
1	4	Gswer	Cwerwerw	China
2	6	Cwerwer	Hwety	China

insert involving two tables

Firstly, let's create table which involving two tables.

```
create table professorTitle as
select first_name, last_name, Type
from Professors,
     titles,
     Titles_Professors
where ProfessorId = Professors_ProfessorId
  and TitleId = Titles_TitleId;
```

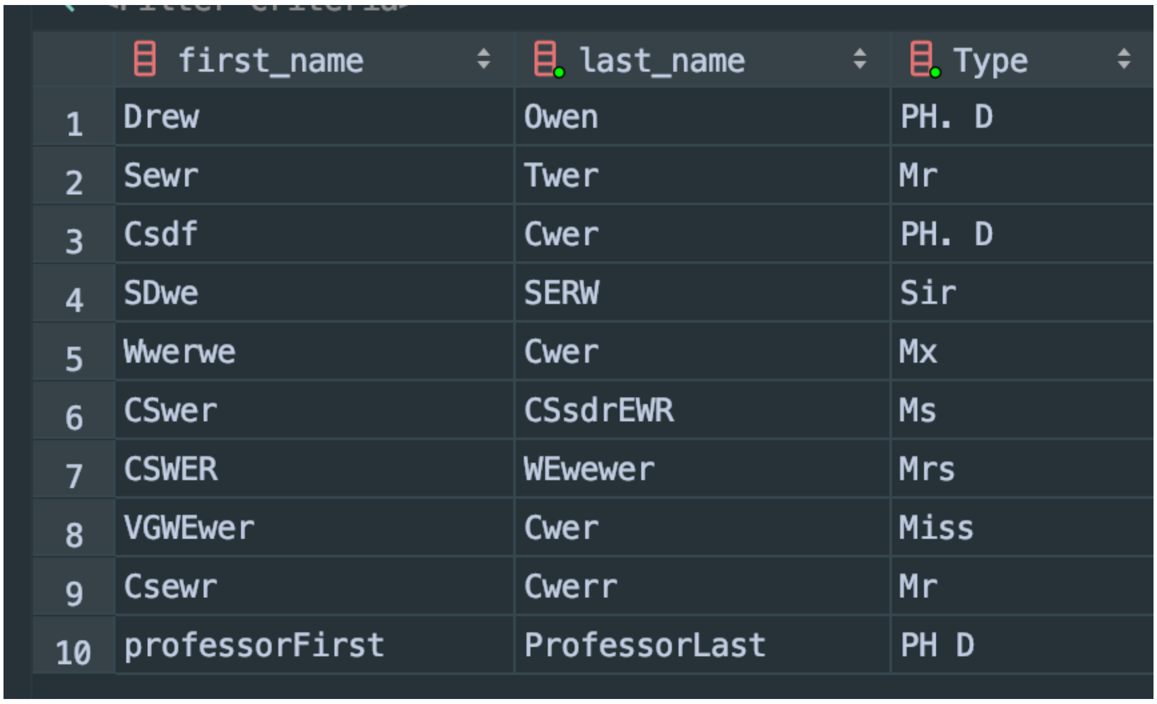

<Filter criteria>

	first_name	last_name	Type
1	Drew	Owen	PH. D
2	Sewr	Twer	Mr
3	Csdf	Cwer	PH. D
4	SDwe	SERW	Sir
5	Wwerwe	Cwer	Mx
6	CSwer	CSsdrEWR	Ms
7	CSWER	WEwewer	Mrs
8	VGWEwer	Cwer	Miss
9	Csewr	Cwerr	Mr

Then, insert to table by using SQL:

```
create table professorTitle as
select first_name, last_name, Type
from Professors,
    titles,
    Titles_Professors
where ProfessorId = Professors_ProfessorId
  and TitleId = Titles_TitleId;
```

After insert, the table become to:



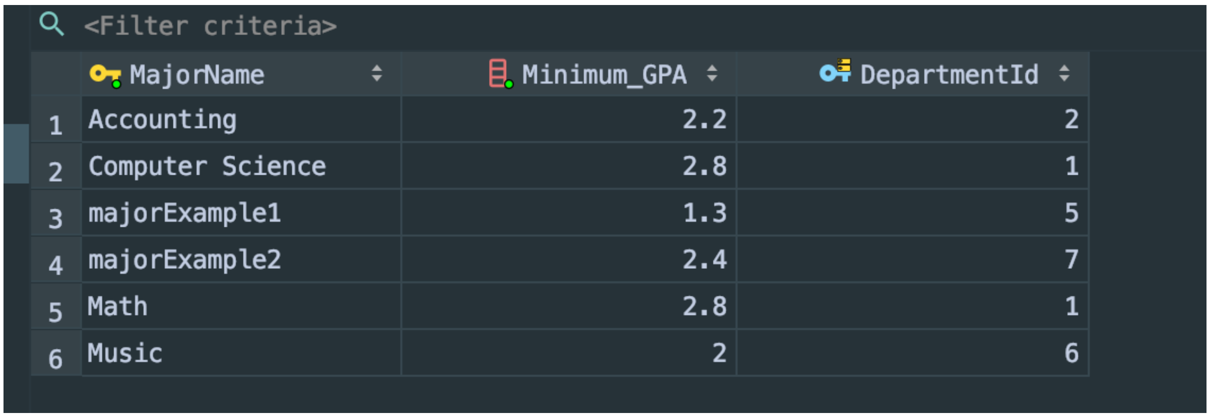
	first_name	last_name	Type
1	Drew	Owen	PH. D
2	Sewr	Twer	Mr
3	Csdf	Cwer	PH. D
4	SDwe	SERW	Sir
5	Wwerwe	Cwer	Mx
6	CSwer	CSsdrEWR	Ms
7	CSWER	WEwewer	Mrs
8	VGWEwer	Cwer	Miss
9	Csewr	Cwerr	Mr
10	professorFirst	ProfessorLast	PH D

- Delete: delete one tuple or a set of tuples: from one table, from multiple tables.

From one table:

My requirement is deleting all about music major information.

Table before delete:



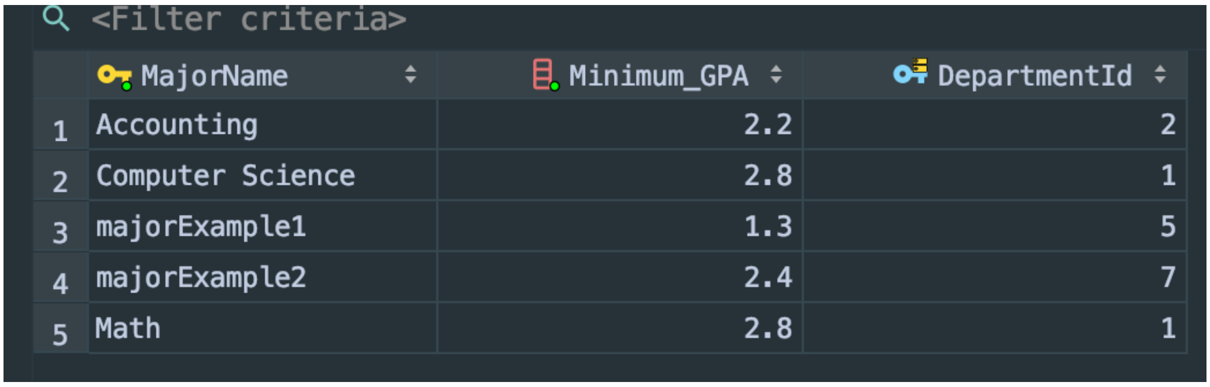
<Filter criteria>

	MajorName	Minimum_GPA	DepartmentId
1	Accounting	2.2	2
2	Computer Science	2.8	1
3	majorExample1	1.3	5
4	majorExample2	2.4	7
5	Math	2.8	1
6	Music	2	6

Mysql:

```
delete from Major
where MajorName = "music";
```

Table after delete:



<Filter criteria>

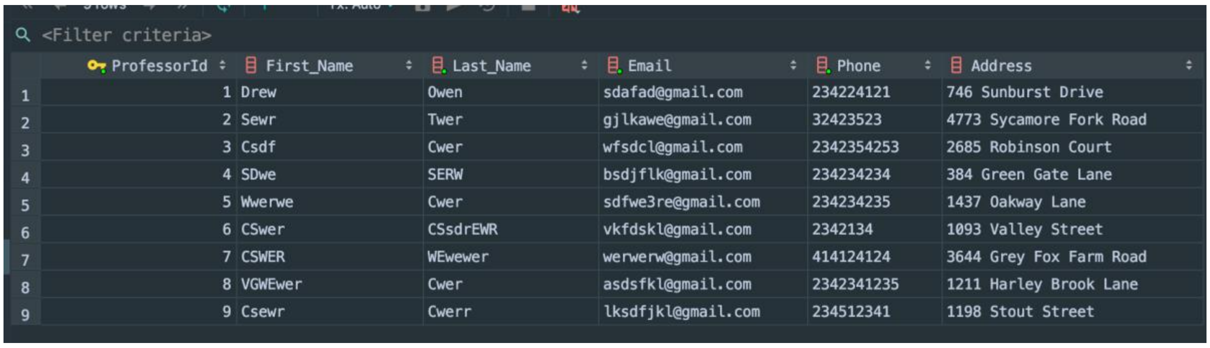
	MajorName	Minimum_GPA	DepartmentId
1	Accounting	2.2	2
2	Computer Science	2.8	1
3	majorExample1	1.3	5
4	majorExample2	2.4	7
5	Math	2.8	1

From multiple table to delete:

My requirement is delete all professors Title is PH.

D

Table before delete:



<Filter criteria>

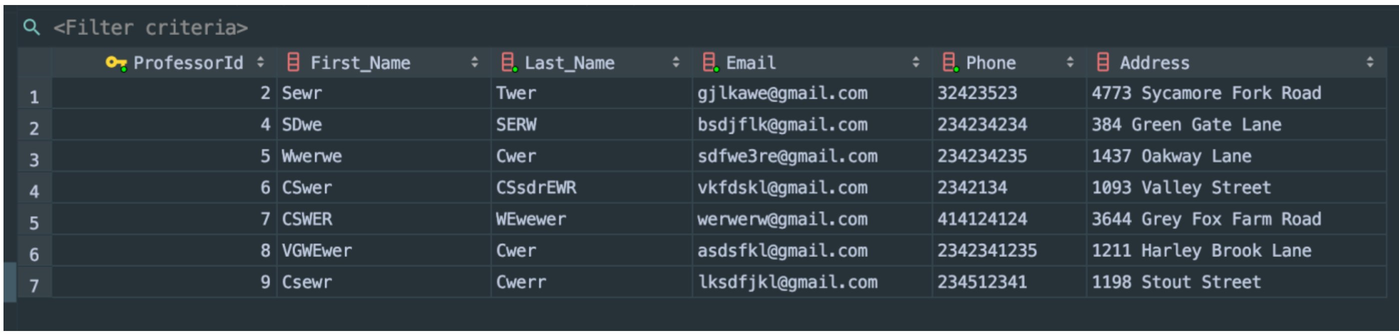
	ProfessorId	First_Name	Last_Name	Email	Phone	Address
1	1	Drew	Owen	sdafad@gmail.com	234224121	746 Sunburst Drive
2	2	Sewr	Twer	gjlkawe@gmail.com	32423523	4773 Sycamore Fork Road
3	3	Csdf	Cwer	wfsdcl@gmail.com	2342354253	2685 Robinson Court
4	4	SDwe	SERW	bsdjflk@gmail.com	234234234	384 Green Gate Lane
5	5	Wwerwe	Cwer	sdfwe3re@gmail.com	234234235	1437 Oakway Lane
6	6	CSwer	CSsdrEWR	vkfdskl@gmail.com	2342134	1093 Valley Street
7	7	CSWER	WEwewer	werwerw@gmail.com	414124124	3644 Grey Fox Farm Road
8	8	VGWEwer	Cwer	asdsfkl@gmail.com	2342341235	1211 Harley Brook Lane
9	9	Csewr	Cwerr	lksdfjkl@gmail.com	234512341	1198 Stout Street

MySQL:

```
delete
from Professors
```

```
where ProfessorId in (select professors_ProfessorId
                      from Titles_Professors
                      where Titles_TitleId = (select TitleId from Titles where type
= "PH. D"));
```

Table after delete:



<Filter criteria>

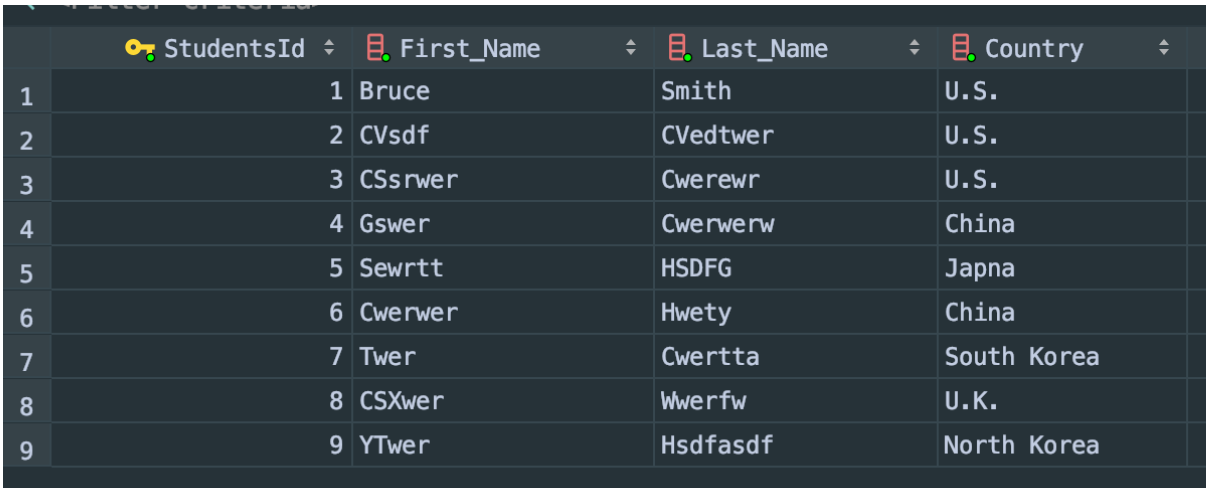
	ProfessorId	First_Name	Last_Name	Email	Phone	Address
1	2	Sewr	Twer	gjlkawe@gmail.com	32423523	4773 Sycamore Fork Road
2	4	SDwe	SERW	bsdjflk@gmail.com	234234234	384 Green Gate Lane
3	5	Wwerwe	Cwer	sdfwe3re@gmail.com	234234235	1437 Oakway Lane
4	6	CSwer	CSsdrEWR	vkfdskl@gmail.com	2342134	1093 Valley Street
5	7	CSWER	WEwewer	werwerw@gmail.com	414124124	3644 Grey Fox Farm Road
6	8	VGWEwer	Cwer	asdsfkl@gmail.com	2342341235	1211 Harley Brook Lane
7	9	Csewr	Cwerr	lksdfjkl@gmail.com	234512341	1198 Stout Street

- Update: update one tuple or a set of tuples: from one table, from multiple tables.

From one table:

My requirement is to change student Bruce Smith's country from U.S. to U.K.

Table before update:

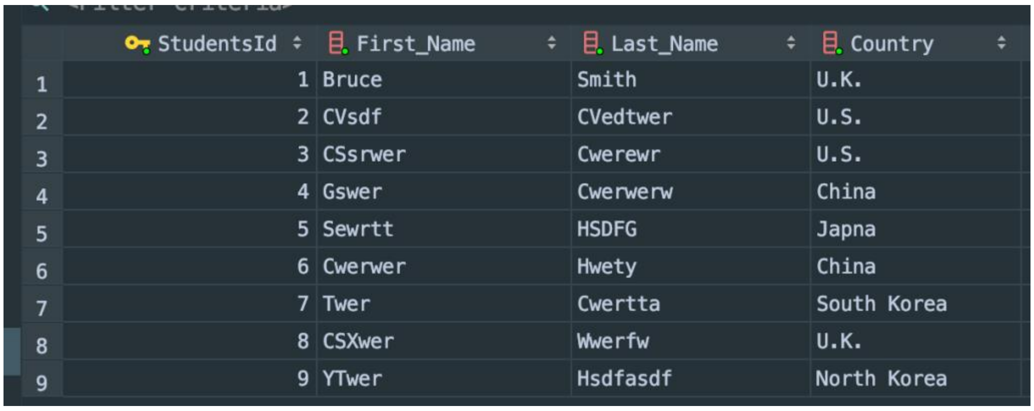


	StudentsId	First_Name	Last_Name	Country
1	1	Bruce	Smith	U.S.
2	2	CVsdf	CVedtwer	U.S.
3	3	CSsrwer	Cwerewr	U.S.
4	4	Gswer	Cwerwerw	China
5	5	Sewrtt	HSDFG	Japna
6	6	Cwerwer	Hwety	China
7	7	Twer	Cwertta	South Korea
8	8	CSXwer	Wwerfw	U.K.
9	9	YTwer	Hsdfasdf	North Korea

MySQL:

```
update students
set country = "U.K."
where First_Name = "Bruce"
and Last_Name = "Smith";
```

Table after update:

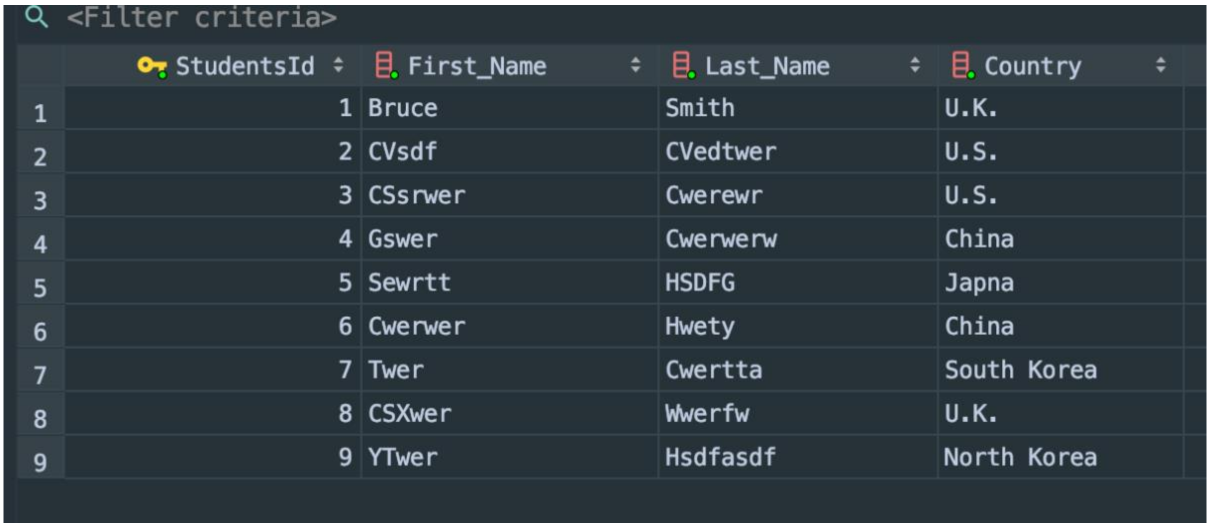


	StudentsId	First_Name	Last_Name	Country
1	1	Bruce	Smith	U.K.
2	2	CVsdf	CVedtwer	U.S.
3	3	CSsrwer	Cwerewr	U.S.
4	4	Gswer	Cwerwerw	China
5	5	Sewrtt	HSDFG	Japna
6	6	Cwerwer	Hwety	China
7	7	Twer	Cwertta	South Korea
8	8	CSXwer	Wwerfw	U.K.
9	9	YTwer	Hsdfasdf	North Korea

From multiple table to update:

My requirement is updating all computer science students' country to Canada.

Table before update:



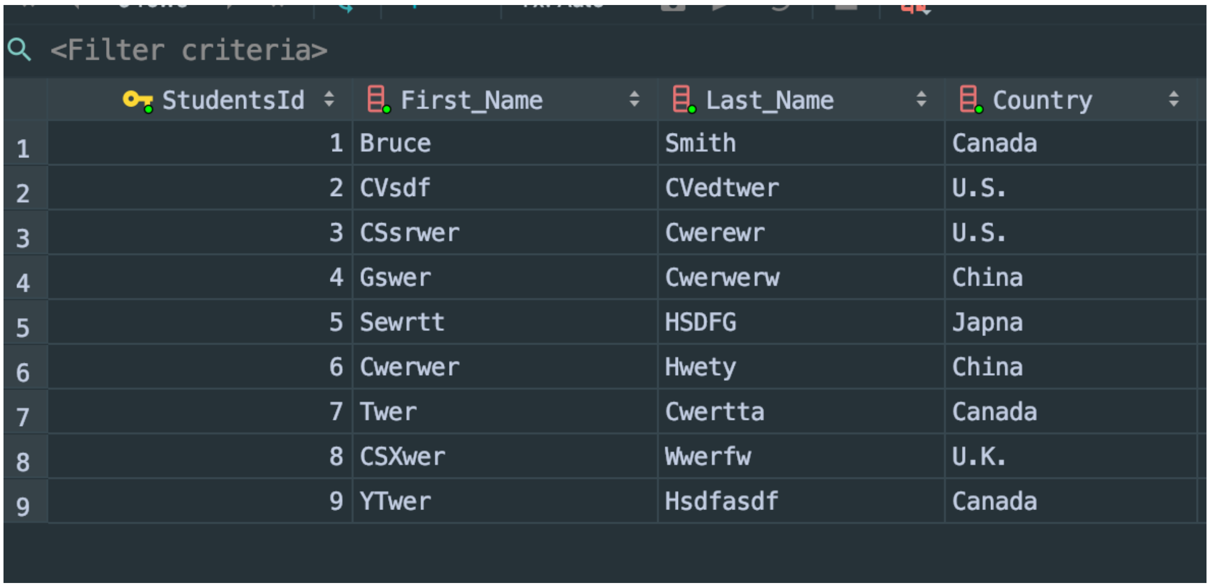
<Filter criteria>

	StudentsId	First_Name	Last_Name	Country
1	1	Bruce	Smith	U.K.
2	2	CVsdf	CVedtwer	U.S.
3	3	CSsrwer	Cwerewr	U.S.
4	4	Gswer	Cwerwerw	China
5	5	Sewrtt	HSDFG	Japna
6	6	Cwerwer	Hwety	China
7	7	Twer	Cwertta	South Korea
8	8	CSXwer	Wwerfw	U.K.
9	9	YTwer	Hsdfasdf	North Korea

MySQL:

```
update Students
set Country = "Canada"
where StudentsId in (select Students_StudentId
                     from Students_Major
                     where Major_MajorName = "Computer Science");
```

Table after update:



<Filter criteria>

	StudentsId	First_Name	Last_Name	Country
1	1	Bruce	Smith	Canada
2	2	CVsdf	CVedtwer	U.S.
3	3	CSsrwer	Cwerewr	U.S.
4	4	Gswer	Cwerwerw	China
5	5	Sewrtt	HSDFG	Japna
6	6	Cwerwer	Hwety	China
7	7	Twer	Cwertta	Canada
8	8	CSXwer	Wwerfw	U.K.
9	9	YTwer	Hsdfasdf	Canada

- Create View:

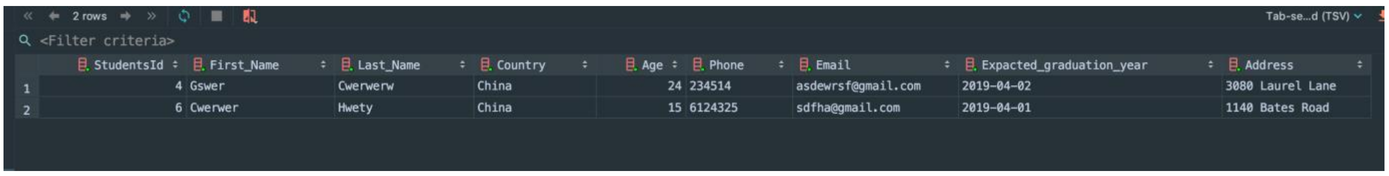
- based on one relation and more than one relation:

One relation view:

If I want to create a view to show all Chinese students, I will write MySQL below:

```
create view CHIN_students_info as
select *
from students
where Country = "China";
```

output:



2 rows Tab-se...d (TSV)
<Filter criteria>

	StudentsId	First_Name	Last_Name	Country	Age	Phone	Email	Expacted_graduation_year	Address
1	4	Gswer	Cwerwerw	China	24	234514	asdewrsf@gmail.com	2019-04-02	3080 Laurel Lane
2	6	Cwerwer	Hwety	China	15	6124325	sdfha@gmail.com	2019-04-01	1140 Bates Road

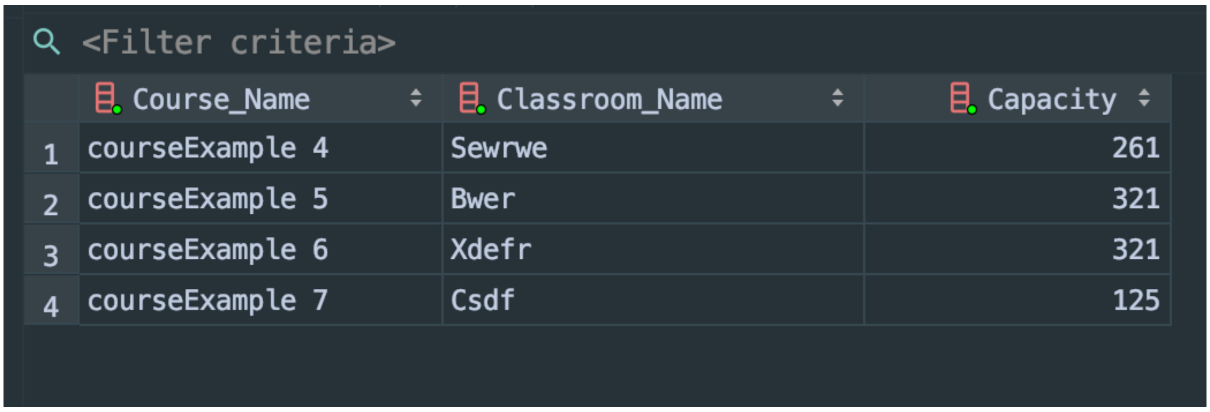
More than one relation view:

If I want to create a view, I called high_capacity_Classroom_Course which would show a courser’s classroom capacity can contain more than 100 students.

MySQL:

```
create view High_capacity_Classroom_Course as
    select Course_Name, Classroom_Name, Capacity from course,
Classroom, Classroom_Course
where courseId = CourseCourseId
and Classromm_ClassroomId = ClassroomId
and Capacity > 100;
```

Output:



<Filter criteria>

	Course_Name	Classroom_Name	Capacity
1	courseExample 4	Sewrwe	261
2	courseExample 5	Bwer	321
3	courseExample 6	Xdefr	321
4	courseExample 7	Csdf	125

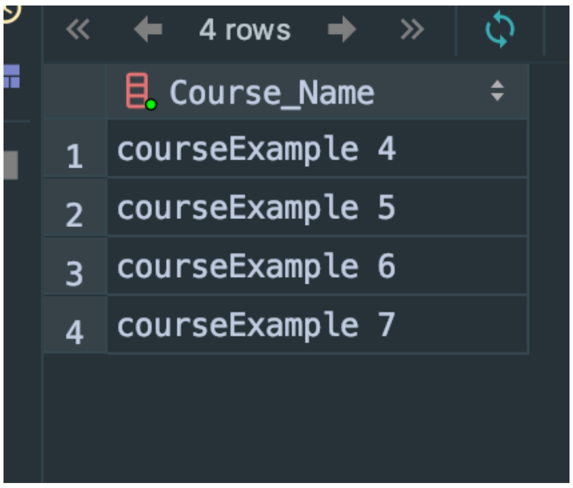
- operate on View (i.e., select, insert, delete, update,..)

I'm going to operate my view I created last question high_capacity_Classroom_Course.

Select VIEW:

```
select Course_Name from high_capacity_classroom_course;
```

Output:


4 rows

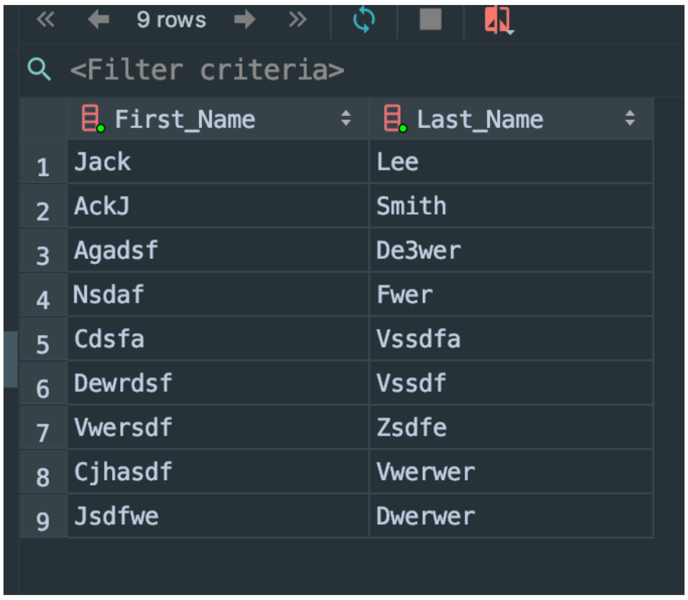
```
  Course_Name
1 courseExample 4
2 courseExample 5
3 courseExample 6
4 courseExample 7
```

Insert VIEW:

First, let's create another view:

```
create view Advisor_name as
    select First_Name, Last_Name from Advisor;
```

Table before insert:


9 rows

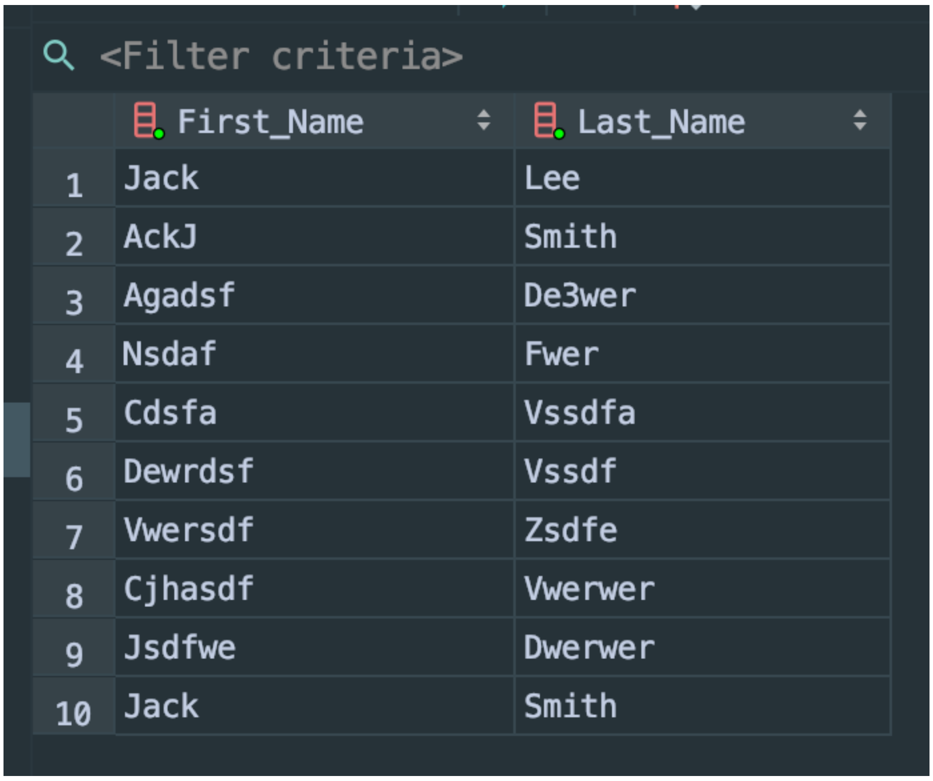
<Filter criteria>

	First_Name	Last_Name
1	Jack	Lee
2	AckJ	Smith
3	Agadsf	De3wer
4	Nsdaf	Fwer
5	Cdsfa	Vssdfa
6	Dewrdsf	Vssdf
7	Vwersdf	Zsdfe
8	Cjhasdf	Vwerwer
9	Jsdfwe	Dwerwer

Insert view MySQL:

```
insert into advisor_name (First_Name, Last_Name) VALUES
("Jack", "Smith");
```

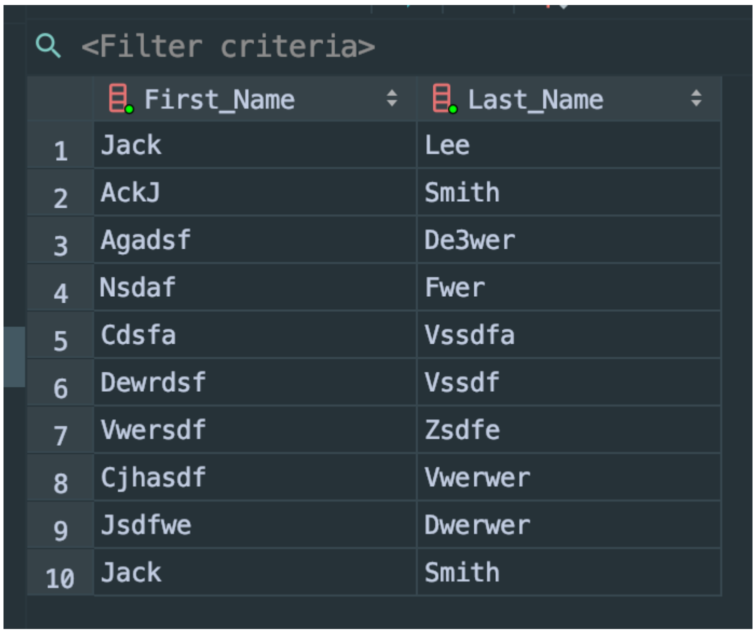
Table after insert:


<Filter criteria>

	First_Name	Last_Name
1	Jack	Lee
2	AckJ	Smith
3	Agadsf	De3wer
4	Nsdaf	Fwer
5	Cdsfa	Vssdfa
6	Dewrdsf	Vssdf
7	Vwersdf	Zsdfe
8	Cjhasdf	Vwerwer
9	Jsdfwe	Dwerwer
10	Jack	Smith

Delete VIEW:

Table before delete:



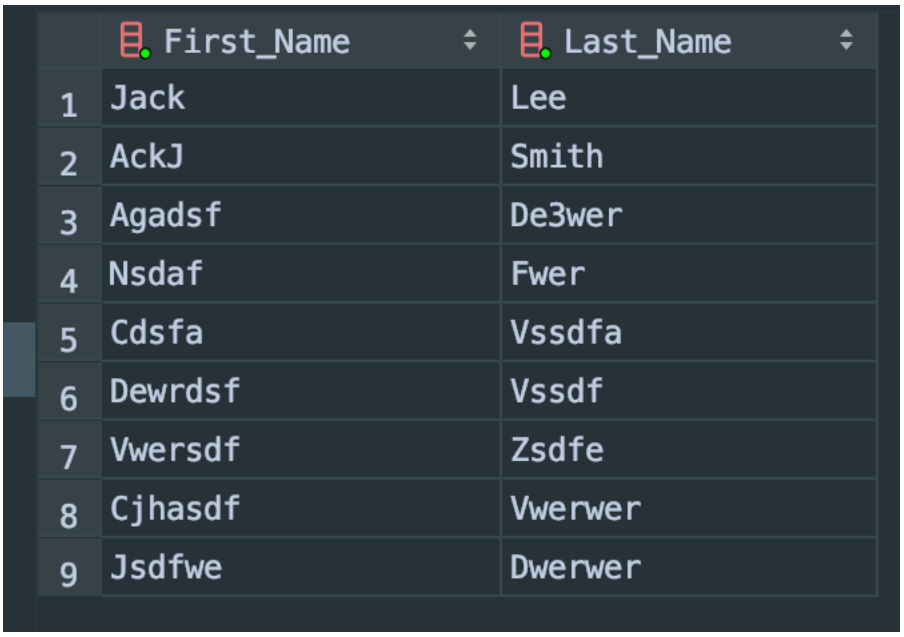
<Filter criteria>

	First_Name	Last_Name
1	Jack	Lee
2	AckJ	Smith
3	Agadsf	De3wer
4	Nsdaf	Fwer
5	Cdsfa	Vssdfa
6	Dewrdsf	Vssdf
7	Vwersdf	Zsdfe
8	Cjhasdf	Vwerwer
9	Jsdfwe	Dwerwer
10	Jack	Smith

MySQL:

```
delete from advisor_name
where First_Name = "Jack"
and Last_Name = "Smith";
```

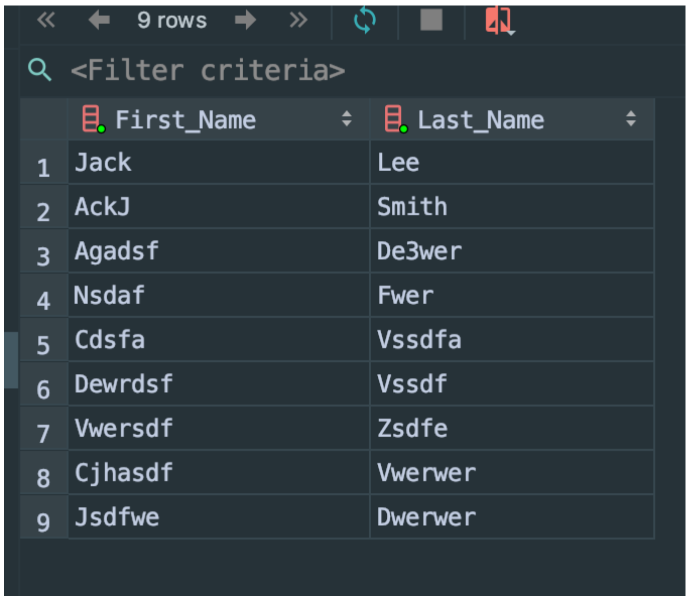
Table after delete:



	First_Name	Last_Name
1	Jack	Lee
2	AckJ	Smith
3	Agadsf	De3wer
4	Nsdaf	Fwer
5	Cdsfa	Vssdfa
6	Dewrdsf	Vssdf
7	Vwersdf	Zsdfe
8	Cjhasdf	Vwerwer
9	Jsdfwe	Dwerwer

Update VIEW:

Table before update:



9 rows

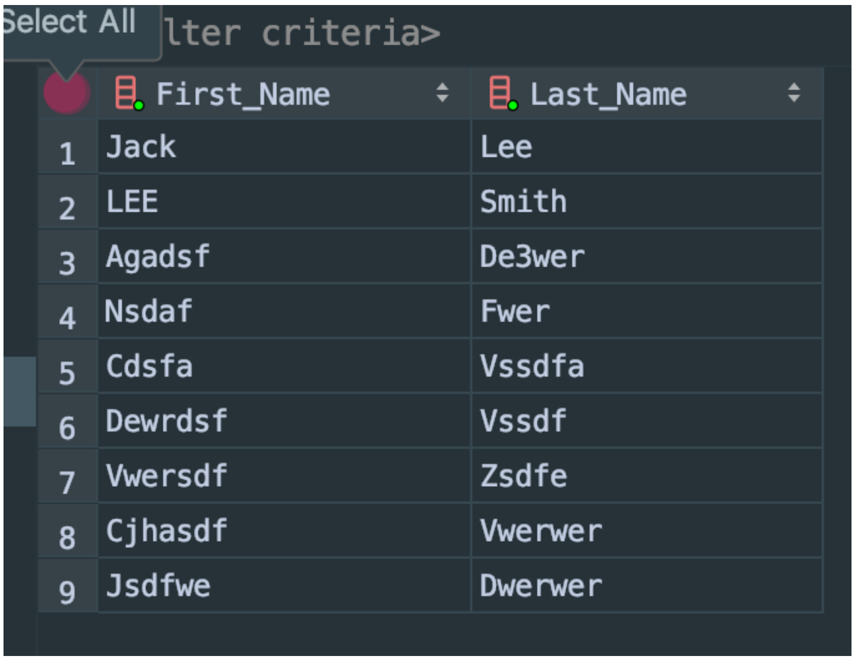
<Filter criteria>

	First_Name	Last_Name
1	Jack	Lee
2	AckJ	Smith
3	Agadsf	De3wer
4	Nsdaf	Fwer
5	Cdsfa	Vssdfa
6	Dewrdsf	Vssdf
7	Vwersdf	Zsdfe
8	Cjhasdf	Vwerwer
9	Jsdfwe	Dwerwer

SQL:

```
update advisor_name set First_Name = "LEE"
where Last_Name = "Smith";
```

Table after update:



	First_Name	Last_Name
1	Jack	Lee
2	LEE	Smith
3	Agadsf	De3wer
4	Nsdaf	Fwer
5	Cdsfa	Vssdfa
6	Dewrdsf	Vssdf
7	Vwersdf	Zsdfe
8	Cjhasdf	Vwerwer
9	Jsdfwe	Dwerwer

TRIGGER

B) Also, create at least 4 different practical/useful triggers (written in MySQL) for your database to perform the following tasks:

- enforcing referential integrity

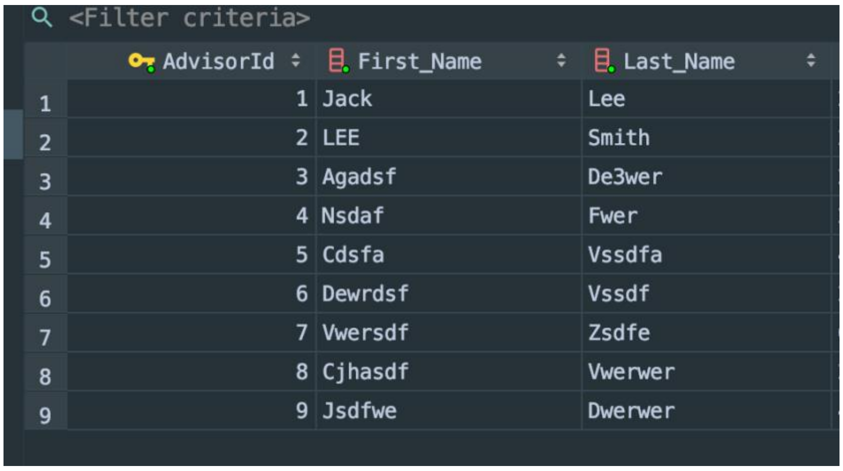
I create a trigger Advisor_delete. If user delete department, then the department advisor also will delete automatically.

```
Delimiter $$
create trigger Advisor_delete after delete on Department
   for each row
   begin
      delete from Advisor
```

```
        where AdvisorId = old.AdvisorId;
    end $$
```

Let’s check if it is work.

Advisor table:



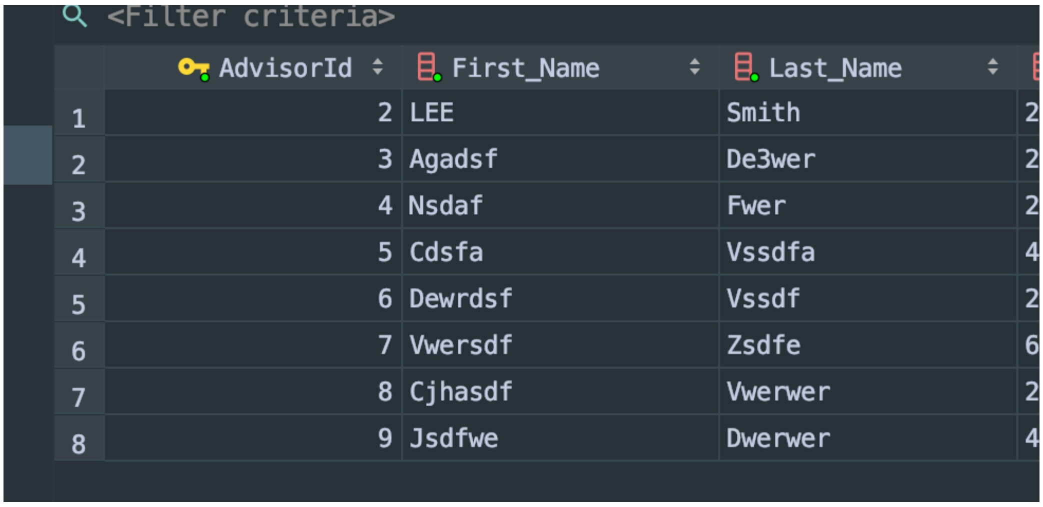
<Filter criteria>

	AdvisorId	First_Name	Last_Name
1	1	Jack	Lee
2	2	LEE	Smith
3	3	Agadsf	De3wer
4	4	Nsdaf	Fwer
5	5	Cdsfa	Vssdfa
6	6	Dewrdsf	Vssdf
7	7	Vwersdf	Zsdfe
8	8	Cjhasdf	Vwerwer
9	9	Jsdfwe	Dwerwer

After I run MySQL below to delete apartment:

```
delete from Department
where DepartmentId = 1;
```

The advisor table become to:



<Filter criteria>

	AdvisorId	First_Name	Last_Name	
1	2	LEE	Smith	2
2	3	Agadsf	De3wer	2
3	4	Nsdaf	Fwer	2
4	5	Cdsfa	Vssdfa	4
5	6	Dewrdsf	Vssdf	2
6	7	Vwersdf	Zsdfe	6
7	8	Cjhasdf	Vwerwer	2
8	9	Jsdfwe	Dwerwer	4

So, the trigger successfully deletes the CAS advisor automatically.

I also create Manage_delete trigger, when user delete department, the trigger will automatically delete the department manager.

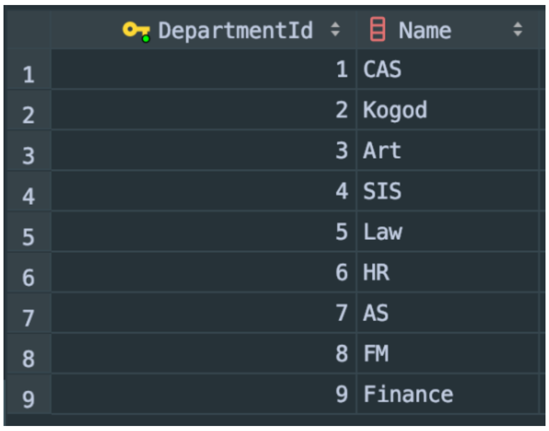
```
Delimiter $$
create trigger Manager_delete after delete on Department
    for each row
    begin
        delete from Manager
            where ManagerId = old.ManagerId;
    end $$
```

Since the method of the trigger is the same, I didn’t take the screenshot for this trigger.

- enforcing attribute domain constraints

I’m going to make trigger that constraints college’s department.

Right now, my database department is below:



	DepartmentId	Name
1	1	CAS
2	2	Kogod
3	3	Art
4	4	SIS
5	5	Law
6	6	HR
7	7	AS
8	8	FM
9	9	Finance

I don’t want to user add any other major to a new department.

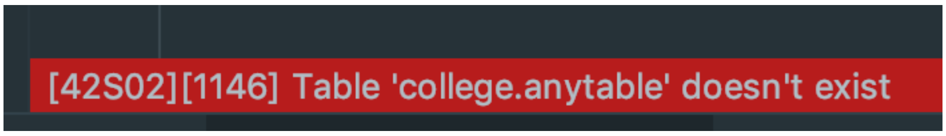
So, the trigger should be:

```
Delimiter $$
Create trigger Department_domain_checking before insert on
Major
For each row
Begin
Declare temp int; set temp=0;
Select count(*) into temp
From Department where
Department.DepartmentId=new.DepartmentId;
If temp=0 then
Insert into mylog values('Invalid Major');
Insert into anytable values('This major is not offered'); End if;
End; $$
```

Let’s test (I create a new departmentId 15):

```
insert into major (MajorName, Minimum_GPA, DepartmentId)
values ("test",2.3,15);
```

Then the error pop up:



```
[42S02][1146] Table 'college.anytable' doesn't exist
```

We are not allowed to add major to a new department anymore (15 is new department ID). The trigger is successfully prevented that happen.

- creating database log

For this trigger, I’m going to create new table called log.

```
Create table StudentLog (message varchar(70));
```

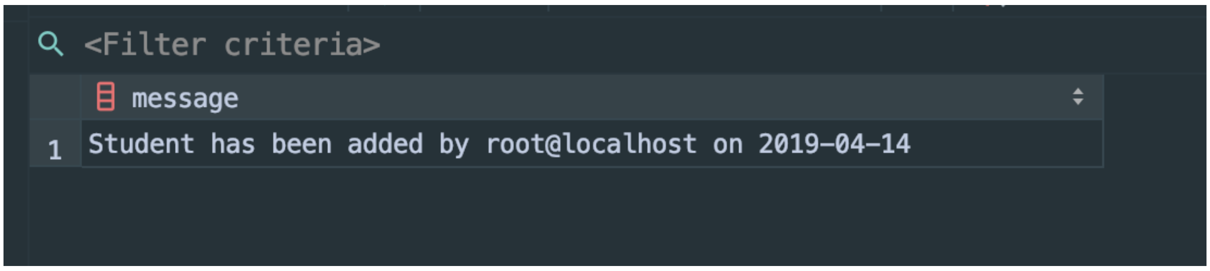
Trigger:

```
Delimiter $$
create trigger Student_Log after insert on students
   for each row
   begin
      insert into StudentLog values(concat('Student has been added by
',current_user(), ' on ',current_date()));
   end;
   $$
```

Let’s active by using:

```
insert into students (First_Name, Last_Name, Country, Age,
Phone, Email, Expacted_graduation_year, Address)
values ("Bruce", "Guo", "Japna", 21, "20234234",
"asdfaea@gmail.com", "2021-04-09", "addresss test 1");
```

Then the studentLog automatically add information below:



```
message
1  Student has been added by root@localhost on 2019-04-14
```

- gathering statistics

I’m going to create student’s age summary trigger.

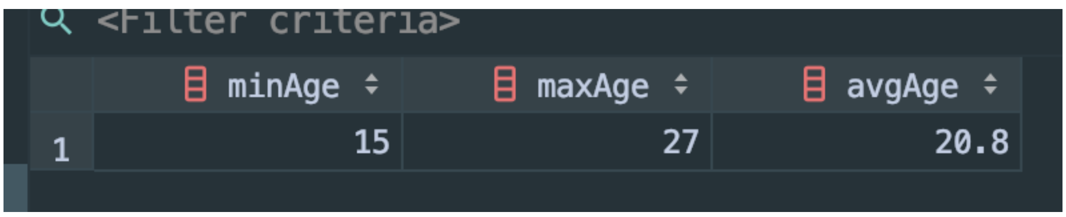
First, create table student_age_summary

```
create table student_age_summary(minAge double, maxAge double, avgAge
double);
```

The trigger code should be:

```
Delimiter $$
create trigger age_sum after insert on Students
for each row
begin
delete from student_age_summary;
insert student_age_summary
select min(age),max(age),avg(age) from students;
end; $$
```

After I insert a new student in student table, the trigger automatically made Age summary for me:



```
   minAge  maxAge  avgAge
1      15      27    20.8
```

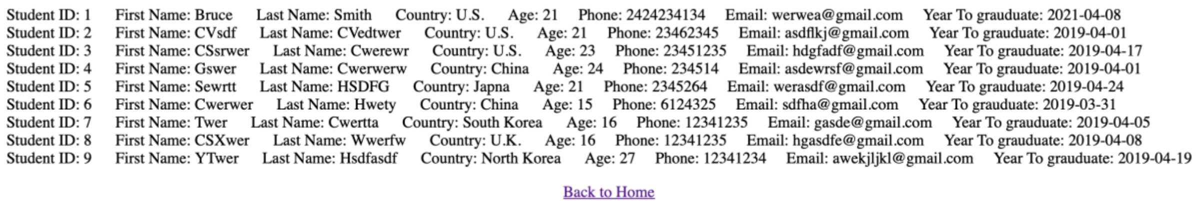
ColdFusion

Use MySQL and ColdFusion to create a Web-based application to enable the user to do the following operations:

Add a record

I made “add student” on web server.

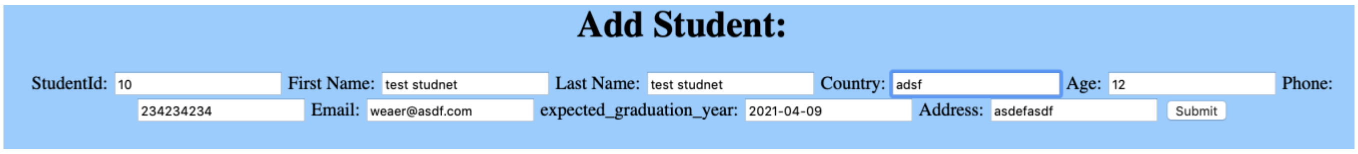
Student table before adding:



```
Student ID: 1   First Name: Bruce    Last Name: Smith      Country: U.S.          Age: 21   Phone: 2424234134   Email: werwea@gmail.com      Year To grauduate: 2021-04-08
Student ID: 2   First Name: CVsdf    Last Name: CVedtwer   Country: U.S.          Age: 21   Phone: 23462345     Email: asdflkj@gmail.com     Year To grauduate: 2019-04-01
Student ID: 3   First Name: CSsrwer  Last Name: Cwerewr    Country: U.S.          Age: 23   Phone: 23451235     Email: hdgfadf@gmail.com     Year To grauduate: 2019-04-17
Student ID: 4   First Name: Gswer    Last Name: Cwerwerw   Country: China         Age: 24   Phone: 234514       Email: asdewrsf@gmail.com    Year To grauduate: 2019-04-01
Student ID: 5   First Name: Sewrtt   Last Name: HSDFG      Country: Japna         Age: 21   Phone: 2345264      Email: werasdf@gmail.com     Year To grauduate: 2019-04-24
Student ID: 6   First Name: Cwerwer  Last Name: Hwety      Country: China         Age: 15   Phone: 6124325      Email: sdfha@gmail.com       Year To grauduate: 2019-03-31
Student ID: 7   First Name: Twer     Last Name: Cwertta    Country: South Korea   Age: 16   Phone: 12341235     Email: gasde@gmail.com       Year To grauduate: 2019-04-05
Student ID: 8   First Name: CSXwer   Last Name: Wwerfw     Country: U.K.          Age: 16   Phone: 12341235     Email: hgasdfe@gmail.com     Year To grauduate: 2019-04-08
Student ID: 9   First Name: YTwer    Last Name: Hsdfasdf   Country: North Korea   Age: 27   Phone: 12341234     Email: awekjljkl@gmail.com   Year To grauduate: 2019-04-19
```

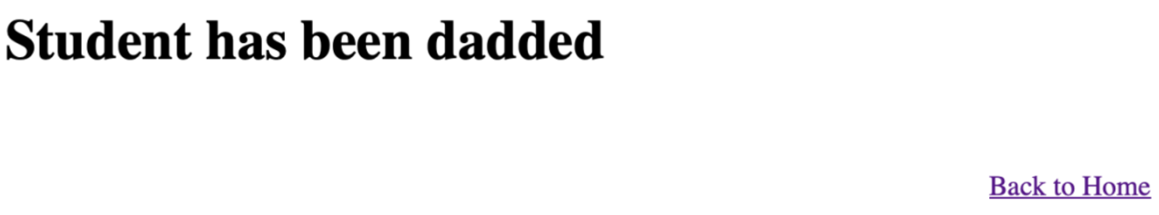
Back to Home

Then, I add student on the web server:



Add Student:

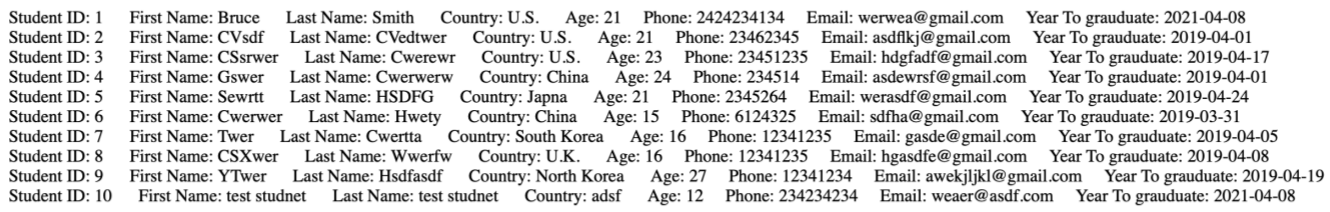
StudentId: 10 First Name: test studnet Last Name: test studnet Country: adsf Age: 12 Phone: 234234234 Email: weaer@asdf.com expected_graduation_year: 2021-04-09 Address: asdefasdf



Student has been dadded

Back to Home

Then, let’s check student table:

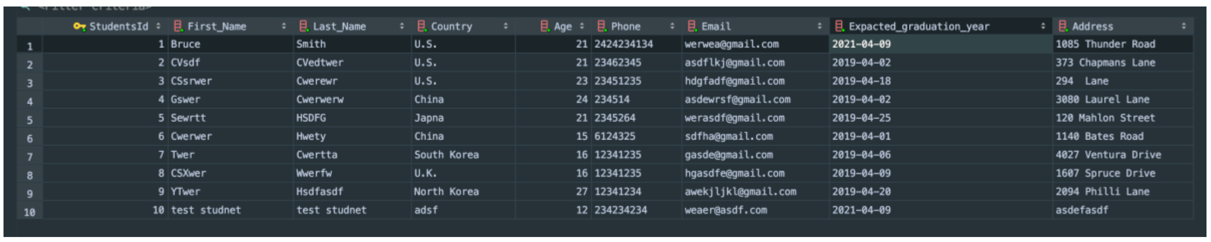


```
Student ID: 1    First Name: Bruce    Last Name: Smith    Country: U.S.    Age: 21    Phone: 2424234134    Email: werwea@gmail.com    Year To grauduate: 2021-04-08
Student ID: 2    First Name: CVsdf    Last Name: CVedtwer    Country: U.S.    Age: 21    Phone: 23462345    Email: asdflkj@gmail.com    Year To grauduate: 2019-04-01
Student ID: 3    First Name: CSsrwer    Last Name: Cwerewr    Country: U.S.    Age: 23    Phone: 23451235    Email: hdgfadf@gmail.com    Year To grauduate: 2019-04-17
Student ID: 4    First Name: Gswer    Last Name: Cwerwerw    Country: China    Age: 24    Phone: 234514    Email: asdewrsf@gmail.com    Year To grauduate: 2019-04-01
Student ID: 5    First Name: Sewrtt    Last Name: HSDFG    Country: Japna    Age: 21    Phone: 2345264    Email: werasdf@gmail.com    Year To grauduate: 2019-04-24
Student ID: 6    First Name: Cwerwer    Last Name: Hwety    Country: China    Age: 15    Phone: 6124325    Email: sdfha@gmail.com    Year To grauduate: 2019-03-31
Student ID: 7    First Name: Twer    Last Name: Cwertta    Country: South Korea    Age: 16    Phone: 12341235    Email: gasde@gmail.com    Year To grauduate: 2019-04-05
Student ID: 8    First Name: CSXwer    Last Name: Wwerfw    Country: U.K.    Age: 16    Phone: 12341235    Email: hgasdfe@gmail.com    Year To grauduate: 2019-04-08
Student ID: 9    First Name: YTwer    Last Name: Hsdfasdf    Country: North Korea    Age: 27    Phone: 12341234    Email: awekjljkl@gmail.com    Year To grauduate: 2019-04-19
Student ID: 10    First Name: test studnet    Last Name: test studnet    Country: adsf    Age: 12    Phone: 234234234    Email: weaer@asdf.com    Year To grauduate: 2021-04-08
```

We can see that student 10 has been added.

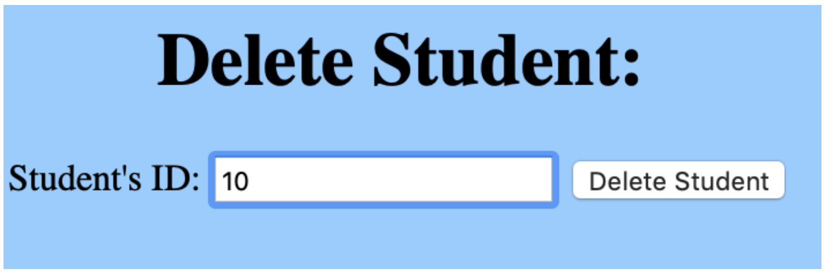
Delete a record

Student table before delete:



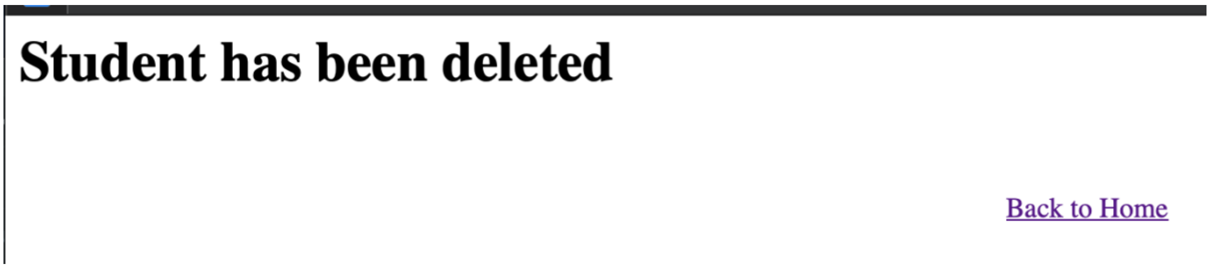
	StudentsId	First_Name	Last_Name	Country	Age	Phone	Email	Expacted_graduation_year	Address
1	1	Bruce	Smith	U.S.	21	2424234134	werwea@gmail.com	2021-04-09	1085 Thunder Road
2	2	CVsdf	CVedtwer	U.S.	21	23462345	asdflkj@gmail.com	2019-04-02	373 Chapmans Lane
3	3	CSsrwer	Cwerewr	U.S.	23	23451235	hdgfadf@gmail.com	2019-04-18	294 Lane
4	4	Gswer	Cwerwerw	China	24	234514	asdewrsf@gmail.com	2019-04-02	3080 Laurel Lane
5	5	Sewrtt	HSDFG	Japna	21	2345264	werasdf@gmail.com	2019-04-25	120 Mahlon Street
6	6	Cwerwer	Hwety	China	15	6124325	sdfha@gmail.com	2019-04-01	1140 Bates Road
7	7	Twer	Cwertta	South Korea	16	12341235	gasde@gmail.com	2019-04-06	4027 Ventura Drive
8	8	CSXwer	Wwerfw	U.K.	16	12341235	hgasdfe@gmail.com	2019-04-09	1607 Spruce Drive
9	9	YTwer	Hsdfasdf	North Korea	27	12341234	awekjljkl@gmail.com	2019-04-20	2094 Philli Lane
10	10	test studnet	test studnet	adsf	12	234234234	weaer@asdf.com	2021-04-09	asdefasdf

Then, I write the studentID in web server:



Delete Student:

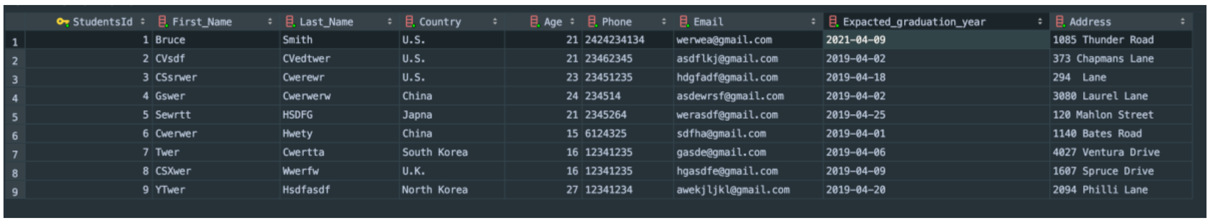
Student's ID: 10



Student has been deleted

Back to Home

Let’s check student table now:



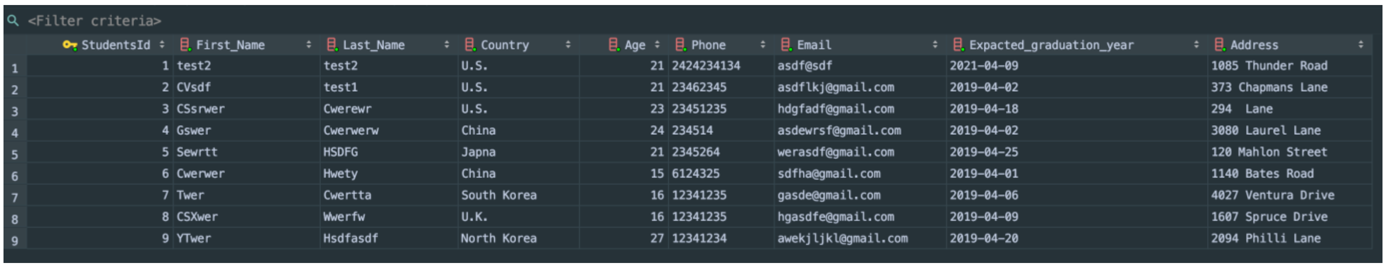
	StudentsId	First_Name	Last_Name	Country	Age	Phone	Email	Expacted_graduation_year	Address
1	1	Bruce	Smith	U.S.	21	2424234134	werwea@gmail.com	2021-04-09	1085 Thunder Road
2	2	CVsdf	CVedtwer	U.S.	21	23462345	asdflkj@gmail.com	2019-04-02	373 Chapmans Lane
3	3	CSsrwer	Cwerewr	U.S.	23	23451235	hdgfadf@gmail.com	2019-04-18	294 Lane
4	4	Gswer	Cwerwerw	China	24	234514	asdewrsf@gmail.com	2019-04-02	3080 Laurel Lane
5	5	Sewrtt	HSDFG	Japna	21	2345264	werasdf@gmail.com	2019-04-25	120 Mahlon Street
6	6	Cwerwer	Hwety	China	15	6124325	sdfha@gmail.com	2019-04-01	1140 Bates Road
7	7	Twer	Cwertta	South Korea	16	12341235	gasde@gmail.com	2019-04-06	4027 Ventura Drive
8	8	CSXwer	Wwerfw	U.K.	16	12341235	hgasdfe@gmail.com	2019-04-09	1607 Spruce Drive
9	9	YTwer	Hsdfasdf	North Korea	27	12341234	awekjljkl@gmail.com	2019-04-20	2094 Philli Lane

Student 10 has been successfully deleted.

Update a record

I made student update ColdFusion.

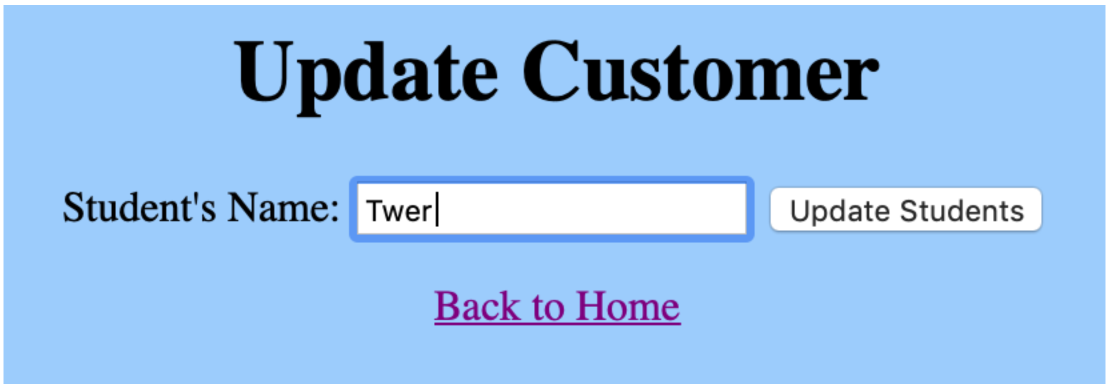
Student table before update:



<Filter criteria>

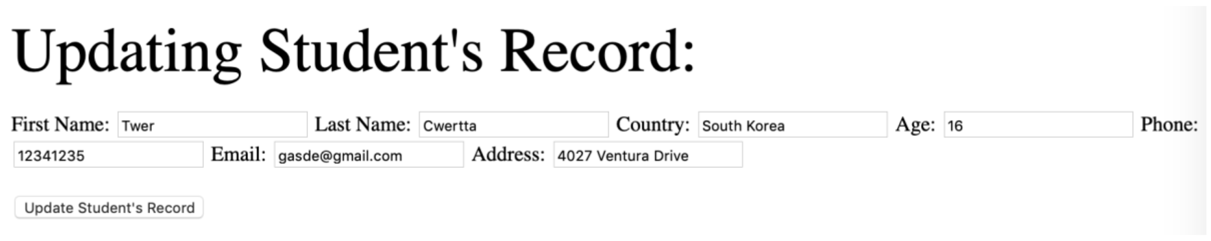
	StudentsId	First_Name	Last_Name	Country	Age	Phone	Email	Expacted_graduation_year	Address
1	1	test2	test2	U.S.	21	2424234134	asdf@sdf	2021-04-09	1085 Thunder Road
2	2	CVsdf	test1	U.S.	21	23462345	asdflkj@gmail.com	2019-04-02	373 Chapmans Lane
3	3	CSsrwer	Cwerewr	U.S.	23	23451235	hdgfadf@gmail.com	2019-04-18	294 Lane
4	4	Gswer	Cwerwerw	China	24	234514	asdewrsf@gmail.com	2019-04-02	3080 Laurel Lane
5	5	Sewrtt	HSDFG	Japna	21	2345264	werasdf@gmail.com	2019-04-25	120 Mahlon Street
6	6	Cwerwer	Hwety	China	15	6124325	sdfha@gmail.com	2019-04-01	1140 Bates Road
7	7	Twer	Cwertta	South Korea	16	12341235	gasde@gmail.com	2019-04-06	4027 Ventura Drive
8	8	CSXwer	Wwerfw	U.K.	16	12341235	hgasdfe@gmail.com	2019-04-09	1607 Spruce Drive
9	9	YTwer	Hsdfasdf	North Korea	27	12341234	awekjljkl@gmail.com	2019-04-20	2094 Philli Lane

Then I put student Twer name to the web server:



Update Customer

Student's Name: Twer

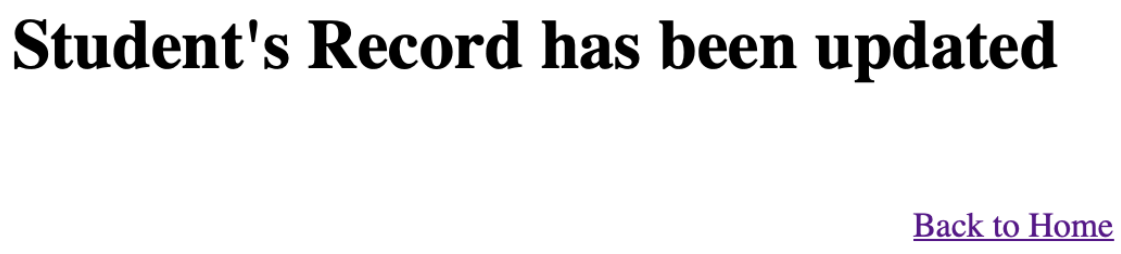


Updating Student's Record:

First Name:	Twer	Last Name:	Cwertta	Country:	South Korea	Age:	16	Phone:	12341235
Email:	gasde@gmail.com	Address:	4027 Ventura Drive						

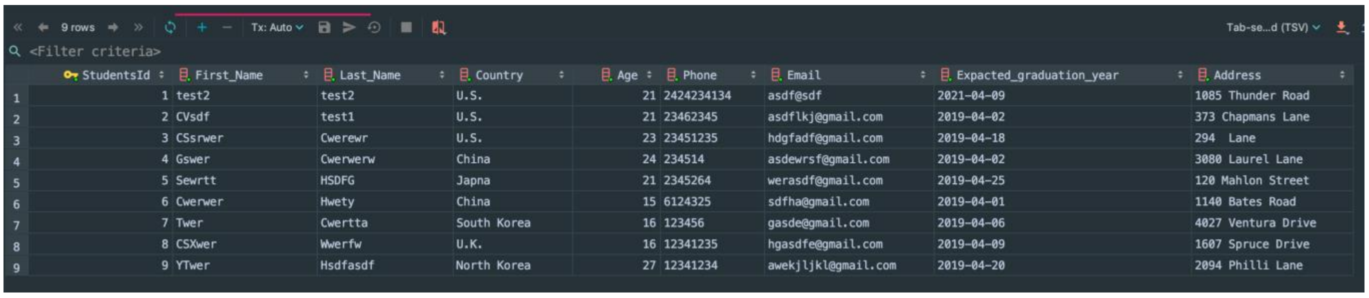
Change the phone number to 123456

After click button:



Student's Record has been updated

Finally, let's check student's table:



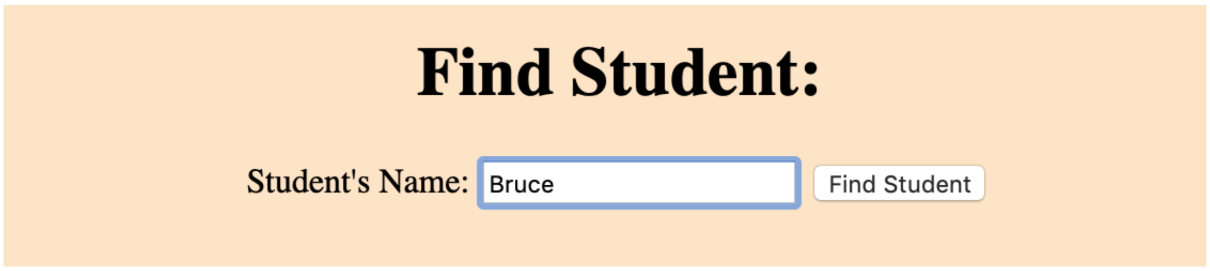
	StudentsId	First_Name	Last_Name	Country	Age	Phone	Email	Expacted_graduation_year	Address
1	1	test2	test2	U.S.	21	2424234134	asdf@sdf	2021-04-09	1085 Thunder Road
2	2	CVsdf	test1	U.S.	21	23462345	asdflkj@gmail.com	2019-04-02	373 Chapmans Lane
3	3	CSsrwer	Cwerewr	U.S.	23	23451235	hdgfadf@gmail.com	2019-04-18	294 Lane
4	4	Gswer	Cwerwerw	China	24	234514	asdewrsf@gmail.com	2019-04-02	3080 Laurel Lane
5	5	Sewrtt	HSDFG	Japna	21	2345264	werasdf@gmail.com	2019-04-25	120 Mahlon Street
6	6	Cwerwer	Hwety	China	15	6124325	sdfha@gmail.com	2019-04-01	1140 Bates Road
7	7	Twer	Cwertta	South Korea	16	123456	gasde@gmail.com	2019-04-06	4027 Ventura Drive
8	8	CSXwer	Wwerfw	U.K.	16	12341235	hgasdfe@gmail.com	2019-04-09	1607 Spruce Drive
9	9	YTwer	Hsdfasdf	North Korea	27	12341234	awekjljkl@gmail.com	2019-04-20	2094 Philli Lane

Student Twer's phone has been updated to 123456.

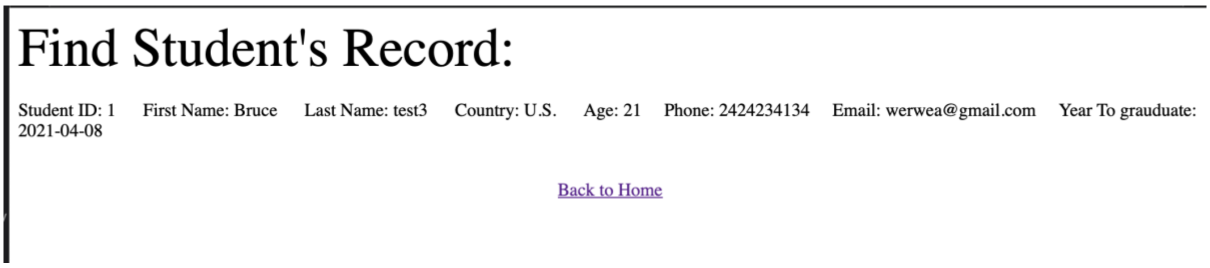
Query (at least 3 select statements on one relation)

The first query: Find students:

As you can see from the screenshot below, you can type student name Bruce here:



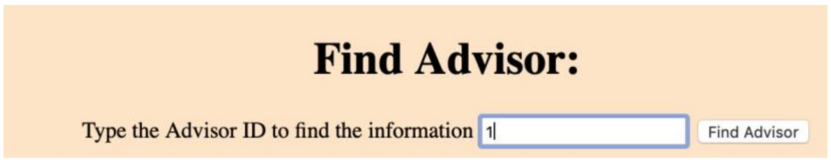
Once you click find student, the page will find the student Bruce’s all information:


Find Student's Record:

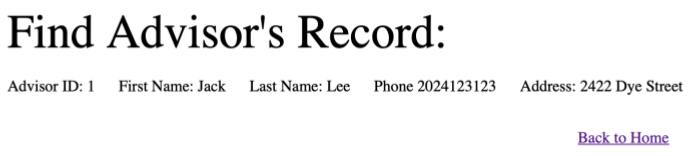
Student ID: 1 First Name: Bruce Last Name: test3 Country: U.S. Age: 21 Phone: 2424234134 Email: werwea@gmail.com Year To grauduate: 2021-04-08

The second query: Find advisor:

As you can see from the screenshot below, you can type advisor ID “1” here:



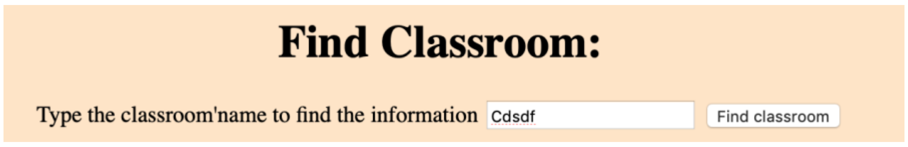
Once you click find advisor, the page will find the advisor’s all information:


Find Advisor's Record:

Advisor ID: 1 First Name: Jack Last Name: Lee Phone 2024123123 Address: 2422 Dye Street

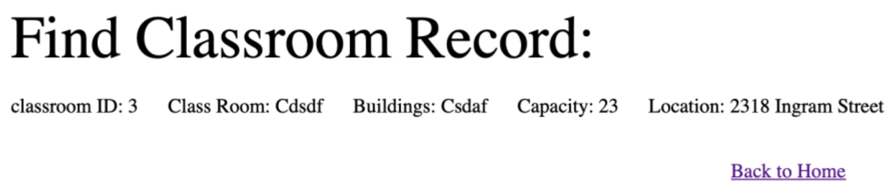
The Third query: Find classroom:

As you can see from the screenshot below, you can type classroom name “Cdsdf” here:


Find Classroom:

Type the classroom'name to find the information Cdsdf

Once you click find advisor, the page will find the classroom’s all information:


Find Classroom Record:

classroom ID: 3 Class Room: Cdsdf Buildings: Csdaf Capacity: 23 Location: 2318 Ingram Street