COURSE REGISTRATION DATABASE

Course: INF 280: Database Systems

Team 2: Lyuben Popov

Alexander Mako

Elizabeth Gibson

Borislav Tsvetanov

Sagyndyk Tukovayev

Final Project Deliverable

Project Specification

Our project represents a university course registration system. We have used AUBG as an example to help us map most of the processes. It proved to be both a challenge and a useful experience because of the many relationships between tables.

We have a *department,* defined by its name, which offers *majors* and *courses.* Every *major* has a unique name and every *course* has a unique id. Moreover, *courses* count for the *major.* The *course* is offered in different *sections,* where one *course* might be offered in several *sections. Sections* have a distinct id. Our model also has *students,* with student id. *Students* can enroll in *sections* and declare *majors.* Each *student* also has a *student bill,* where every *student* has only one account. This is a bit outside our domain but we implemented it early and in our belief, it doesn't bring an unnecessary complexity.

Moreover, our model has *instructors,* each with distinct id. *Instructors* are part of *departments* and are connected via the department name. Furthermore, *instructors* teach *sections* and one *instructor* might *several sections.* Every *student* also has an *advisor,* who might or might not be an *instructor.*

Finally, every *section* is given at a specific *timeslot* and is also assigned a *classroom,* both of which have a unique identifier. We also use a *timetable* to connect the *course, section,* and *timeslot.* Last but not least, every *classroom* is equipped with *equipment,* where the *equipment* has a unique id.

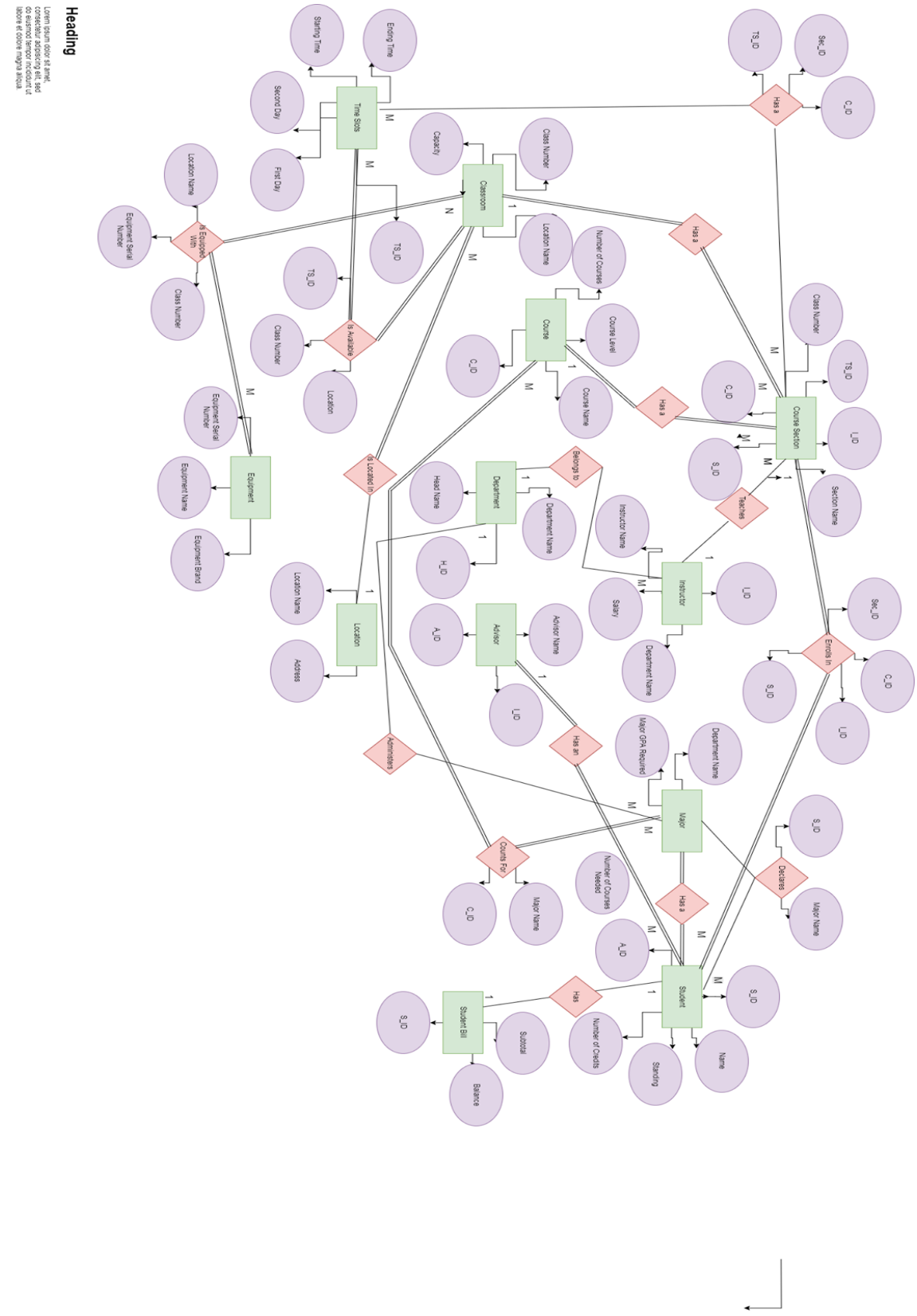
The tables could be seen on the SQL file and we have a picture of the schema.

Contribution: The team met and discussed ideas about the project and jointly decided to do a course registration system. The team constantly updated and reviewed the tables so as to present the most accurate model.

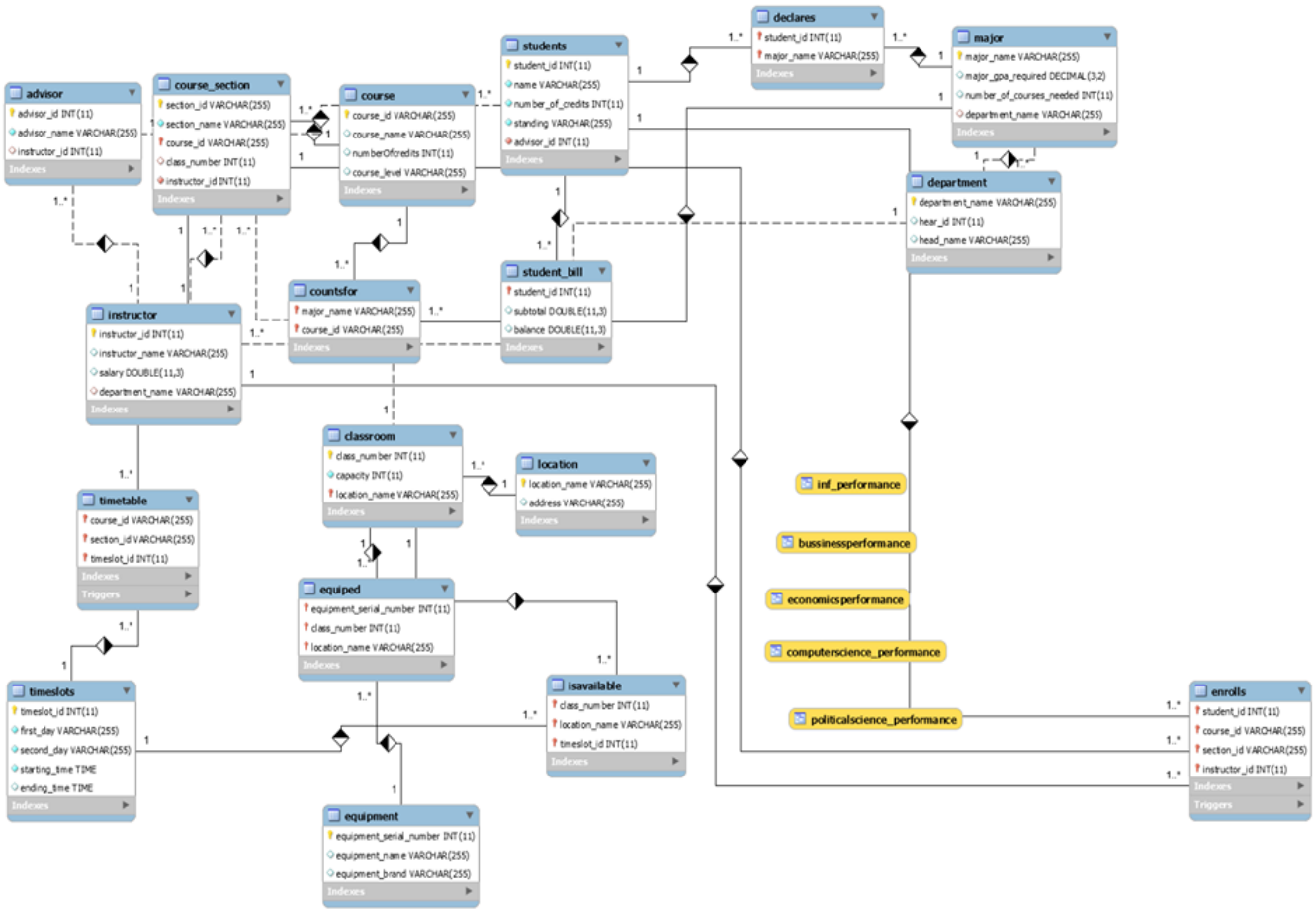
EER Diagram

Here is our EER diagram. The original photo could be seen in the files attached.

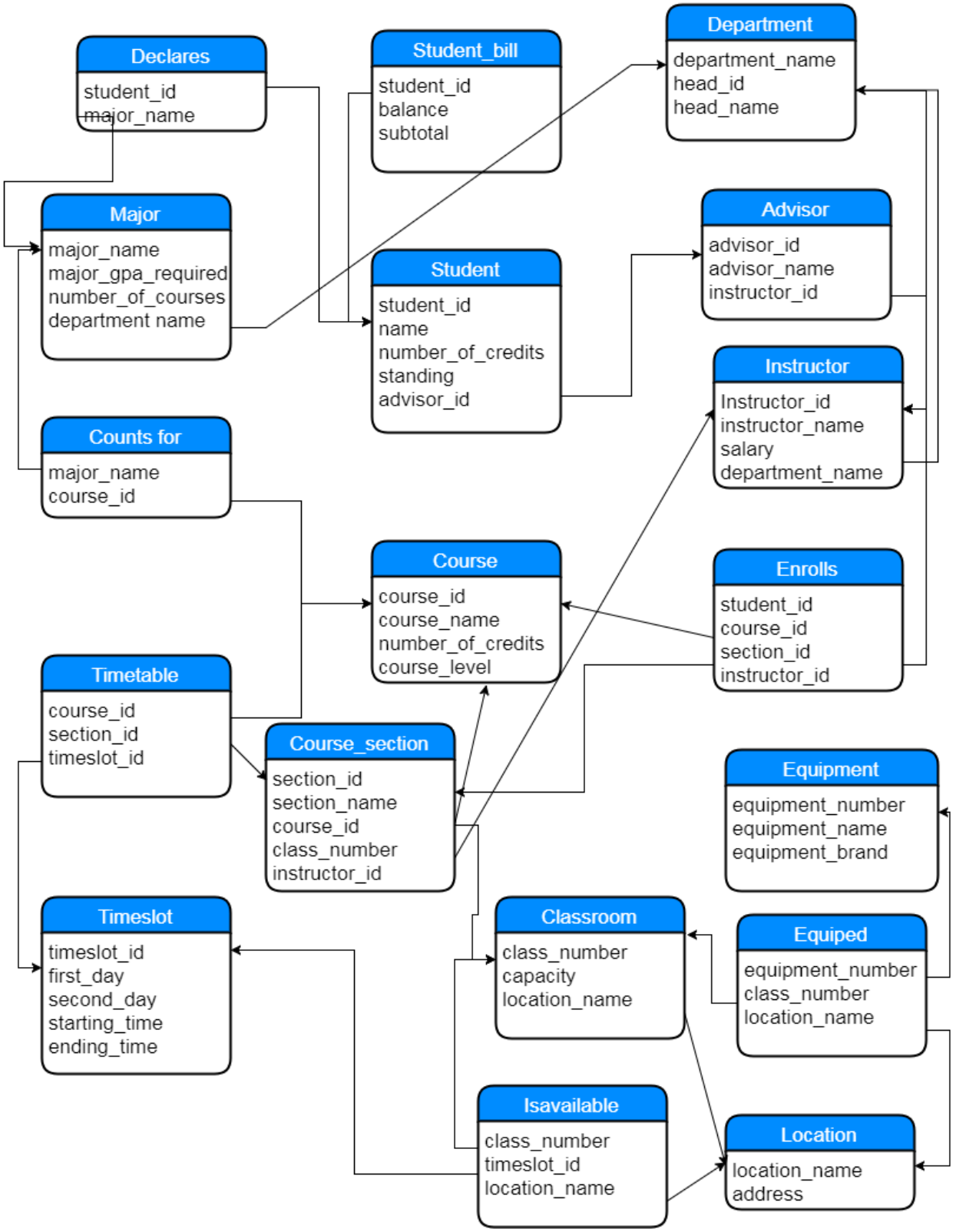
Contribution: Elizabeth and Alex designed it. Lyuben reviewed it and the rest of the team gave feedback.


Heading
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.
Starting Time
Ending Time
Time Slots
Second Day
First Day
Capacity
Class Number
Classroom
Location Name
Is Equipped With
Equipment Serial Number
Class Number
TS_ID
Class Number
Is Available
Location
Number of Courses
Course
Course Level
C_ID
Course Name
Has a
Class Number
TS_ID
C_ID
Course Section
I_ID
S_ID
Section Name
Teaches
Is Located In
Equipment Serial Number
Equipment
Equipment Name
Equipment Brand
Head Name
Department
Department Name
H_ID
Belongs to
Instructor Name
Instructor
I_ID
Salary
Department Name
Location Name
Location
Address
A_ID
Advisor
Advisor Name
I_ID
Administers
Has an
Major GPA Required
Department Name
Major
Counts For
C_ID
Major Name
Number of Courses Needed
A_ID
Has a
Declares
S_ID
Major Name
Has
Student
S_ID
Name
Standing
Number of Credits
Student Bill
S_ID
Subtotal
Balance
Sec_ID
C_ID
Enrolls In
S_ID
I_ID
TS_ID
Sec_ID
C_ID
Has a

Relational Model and Schema



This is our schema. The entire picture is with the rest of the files.



Here is our relational model. Most of the tables are in the 3rd normal form right after the mapping so there was almost no need to do any normalization at all. We

apologize that at some points the relationship arrows might look confusing but the model is a bit complex with many connections present. Finally, the picture of the relational model is present with all the files.

Contribution: Based on the schema, Lyuben created the basic model. The rest of the team gave feedback and corrected any inaccuracies.

Sample Data

We have attached the sample data for the project. Most of the tables have 15 rows of data except the ones, where logic prevents it from having so many, for example, the locations table has only Balkanski and Main Building since they are the only possible lecture locations in AUBG. The data is located in the excel file *data.xlsx*

Contributions: Bobi and Sagyndyk inserted the data and check for its validity. The rest of the team gave feedback.

Queries and Triggers

We give as an example 15 queries and 2 triggers that are tested and work accurately within the database. The queries make use of all the operations we have been studying during the semester such as joins, sub-queries, correlated sub-queries, sets, where, exists, not in. They can be found in the queries text document.

1. **Find the business major that has the highest number of credits accumulated**

```
SELECT name,number_of_credits

FROM students

WHERE number_of_credits =

                (SELECT max(number_of_credits)
```

```
FROM   (SELECT name , student_id,number_of_credits
        FROM students inner join declares
using(student_id)
        INNER JOIN major using (major_name)
        WHERE major.major_name="Bussines")as table1);
```

2. **Find the Computer Science Majors that are in sections where the classroom does not have computers.**

```
SELECT  name, student_id,class_number,section_name
FROM            (SELECT name, student_id
        FROM students inner join declares using (student_id)
        INNER JOIN major using (major_name)
        WHERE major_name="Computer Science")as table1
        INNER JOIN  enrolls using (student_id)
        INNER JOIN course_section using (section_id)
        WHERE class_number  IN(SELECT class_number
                FROM equiped
                WHERE equipment_serial_number
                        NOT IN   (SELECT
                equipment_serial_number
                FROM equipment
        WHERE equipment_name="computer") ) ;
```

3. **Find when are courses conducted and classrooms available**

```
SELECT *
```

```
FROM  (SELECT section_name, timeslot_id,class_number
        FROM course_section INNER JOIN timetable using
        (course_id,section_id)) as table1 INNER JOIN timeslots using
        (timeslot_id)
          WHERE EXISTS( SELECT * FROM
              (SELECT class_number,timeslot_id
               FROM classroom inner join isavailable using(class_number)) as
                                                                    table2
                WHERE table1.timeslot_id=table2.timeslot_id);
```

4. **Find students who have enrolled in courses that double count**

```
SELECT name ,student_id
 FROM students
 WHERE student_id in   (SELECT student_id
                        FROM enrolls
                        WHERE section_id in (SELECT section_id from
course_section
                                  WHERE course_id in (select course.course_id from
                                      course inner join countsfor using (course_id)
                        INNER JOIN major using(major_name)
                        GROUP BY course_id having count(*)>1))) ;
```

5. **Find computer science majors enrolled in sections where classrooms do not have computers**

```
SELECT *
FROM course_section inner join classroom using (class_number)
INNER JOIN equiped using (class_number)
```

```
WHERE class_number not in (SELECT class_number
                           FROM equiped
                           WHERE equipment_serial_number in
                               (SELECT equipment_serial_number
                               FROM equipment
                               WHERE equipment_name="computer"))
                               AND course_id in (SELECT course_id
                                                 FROM course
                                                 WHERE course_id in
                                                 (SELECT course_id
                                                 FROM countsfor
                               WHERE major_name="Computer
Science"));
```

6. **Find classes that have reached their full capacity**

```
SELECT section_name, class_number,capacity from  students inner join enrolls
using (student_id)
INNER JOIN course_section using (section_id)
INNER JOIN classroom using (class_number)
GROUP BY section_name, class_number
HAVING count(*)=capacity;
```

7. **Find students enrolled in sections taught by their advisors**

```
SELECT name,
student_id,course_section.instructor_id,course_section.section_name
```

```
FROM students inner join enrolls using (student_id)
INNER JOIN course_section using (course_id)
WHERE course_section.instructor_id in  (SELECT instructor_id
                                        FROM advisor
                                        WHERE advisor_id =students.advisor_id );
```

8. **Find Junior with second largest amount of credits and no major declared**

```
SELECT name, student_id ,major_name
FROM students left join declares using (student_id)
WHERE standing in ("junior","Junior")
AND number_of_credits in (SELECT max(number_of_credits)
                          FROM students
                          WHERE number_of_credits< (SELECT
                                max(number_of_credits)
                          FROM students))
                          AND major_name is null;
```

9. **Find students that have collisions on their second day**

```
SELECT name,starting_time,ending_time,first_day,second_day
FROM students inner join enrolls using (student_id)
 INNER JOIN course_section using (section_id)
 INNER JOIN course  on course_section.course_id = course.course_id
INNER JOIN timetable on course_section.course_id= timetable.course_id and
course_section.section_id=timetable.section_id
INNER JOIN timeslots on timetable.timeslot_id=timeslots.timeslot_id
```

```
GROUP BY student_id, starting_time , second_day
HAVING count(*)>=2;
```

10.Students that have taken a course that does not count for their major

```
SELECT student_id,name, section_name
FROM students inner join enrolls using (student_id)
INNER JOIN course_section using (section_id)
WHERE course_section.course_id not in  (SELECT countsfor.course_id
                                        FROM countsfor
                                        WHERE major_name=(SELECT major_name
                                                          FROM declares
                                                          WHERE students.student_id=declares.student_id))
                                        HAVING student_id in (SELECT student_id
                                                              FROM declares );
```

11.Find advisors who are not instructors

```
SELECT *
FROM advisor
WHERE instructor_id  is NULL;
```

12.Find students that do not have a major

```
SELECT
student_id,name,course_section.section_name,course.course_name,countsfor
.major_name as counts_for
FROM students inner join enrolls using (student_id)
```

```
INNER JOIN course_section using (section_id)
 INNER JOIN course on course_section.course_id=course.course_id
INNER JOIN countsfor on countsfor.course_id=course.course_id
 LEFT JOIN declares using (student_id)
WHERE declares.major_name is null;
```

13.Find students that have enrolled in more than 3 courses

```
SELECT name, count(*) as Number_of_Course from students inner join enrolls
using (student_id)
 INNER JOIN course_section using (section_id)
 INNER JOIN course  on course_section.course_id = course.course_id
 GRUP BY name
HAVING Number_of_Course > 3;
```

14.Find timetable for students

```
SELECT name,course_name,starting_time,ending_time,first_day,second_day
FROM students inner join enrolls using (student_id)
INNER JOIN course_section using (section_id)
INNER JOIN course  on course_section.course_id = course.course_id
 timetable on course_section.course_id= timetable.course_id and
course_section.section_id=timetable.section_id
INNER JOIN timeslots on timetable.timeslot_id=timeslots.timeslot_id
ORDER BY name
```

15. Find the most on demand major for the current enrollment session – with views

```
SELECT major_name, enrollment_credits,max(enrollment_credits) as top
```

```
FROM (SELECT *
        FROM
        ComputerScience_Performance
        UNION
        SELECT * from BussinessPerformance
        UNION
        SELECT *
        FROM  EconomicsPerformance
        UNION
        SELECT *
        FROM  PoliticalScience_Performance
        UNION
        SELECT *
        FROM  Inf_Performance)s
        HAVING enrollment_credits =top;
```

1. **Trigger – Stop a section from being assigned a timeslot that is not in the set of timeslots, where the assigned classroom is available**

```
CREATE TRIGGER timetable_ai
AFTER INSERT on timetable
FOR EACH row
BEGIN
IF (NEW.timeslot_id not in  (SELECT timeslot_id
                             FROM isavailable
                             WHERE class_number in
```

```
                    ((SELECT class_number
                    FROM classroom
                    WHERE class_number in
                    (SELECT class_number
            FROM course_section INNER JOIN timetable USING
            (section_id)
WHERE section_id=NEW.section_id))))
) then signal sqlstate'45000' set message_text="class is not available at this
time";
end if;
```

Test statement: INSERT INTO timetable(course_id,section_id,timeslot_id) values("ENG 300","eng 300",1);

2. Trigger – Prevent a student from having more than 17 credits

```
DELIMITER //

CREATE trigger enrolls_bi

AFTER INSERT on enrolls

FOR EACH ROW

BEGIN

DECLARE nr_credits int default 0;

SELECT sum(numberOfcredits) into nr_credits

    FROM students inner join enrolls using (student_id)

    INNER JOIN course_section using (section_id)

    INNER JOIN course  on course_section.course_id =
course.course_id

     GROUP BY student_id

     HAVING sum(numberOfcredits)>10;
```

```
IF NR_CREDITS>17  then

SIGNAL sqlstate '45000' set message_text='Maximum number of
credits(17) surpased ' ;

END IF ;

END //

DELIMITER;
```

Contributions: Alex created the queries and triggers, Lyuben provided initial input and tested. The other members gave feedback.

Query Optimization

Because of the relatively small size of the database, it is impossible to show the effect of the actual optimization. In both cases the execution time is (0,00s) but here we show one way of how two of our queries could be optimized.

Query 1

Version 1:

```
select
student_id,name,course_section.section_name,course.course_name,countsfor.major
_name as counts_for

from students inner join enrolls using (student_id)

inner join course_section using (section_id)

 inner join course on course_section.course_id=course.course_id

 inner join countsfor on countsfor.course_id=course.course_id

 left join declares using (student_id)

 where declares.major_name is null;
```

Version 2: optimized

Here we try to find the students that have enrolled for a course that counts for a major but have not declared one. Instead of left joining in the end we can perform the left join in the beginning to first find the students with an undeclared major and then find the courses those students have taken. By doing so we reduce the dataset produced by all the joins and then being left joined with the majors.

```
select
table1.student_id,table1.name,course_section.section_name,course.course_name,co
untsfor.major_name as counts_for

from (select student_id, name  from students left join declares using (student_id)
where declares.major_name is null)as table1

inner join enrolls on table1.student_id=enrolls.student_id

inner join course_section using (section_id)

 inner join course on course_section.course_id=course.course_id

 inner join countsfor on countsfor.course_id=course.course_id;
```

Query 2.

Version 1 :

```
SELECT  name, student_id,class_number,section_name
FROM students INNER JOIN declares using (student_id)
    INNER JOIN major using (major_name)
INNER JOIN enrolls using (student_id)
    INNER JOIN course_section using (section_id)
WHERE class_number IN(SELECT class_number
                      FROM equiped
                      WHERE equipment_serial_number
                      NOT IN   (select equipment_serial_number
                           FROM equipment  where equipment_name="computer") )
                and major_name="Computer Science" ;
```

Execution plan:

Here we try to find the computer science majors enrolled in sections of courses conducted in classes where there are no computers. In order to optimize the query we can firstly try to reduce the intermediate data being joined. Instead of searching for computer science majors after we have joined all the students with all the sections etc. We can start by cutting down the number of students to be joined with the enrollments table and eventually with the course section table by firstly filtering for computer science majors since that is what we are interested in searching for.

```
SELECT  name, student_id,class_number,section_name
FROM
   (SELECT name, student_id
    FROM students inner join declares using (student_id)
    INNER JOIN major using (major_name)
    WHERE MAJOR_name="Computer Science")as table1 -
    INNER JOIN  enrolls using (student_id)
    INNER JOIN course_section using (section_id)
    WHERE class_number
                IN(SELECT class_number
                   FROM equiped
                   WHERE equipment_serial_number
                   NOT IN   (SELECT equipment_serial_number
                        FROM equipment
                      WHERE equipment_name="computer") ) ;
```

Contribution: Alex and Bobi suggested some querry optimizations. The entire team discussed and reach the most appropriate decision.