**Abstract Review System (CAHSI)**

**Database Name:**

f17cs4342team03

**Team Name:**

Data Miners

**Team Members:**

Stephanie Loya

Andres Olivas

Adrian Gonzalez

Joseph Gutierrez

Table of Contents

* 1. Title Page…………………………………………………………………………………………………………………………………………. 1
  2. Table of Contents………………………………………………………………………………………………………………………………2

[1.3 Abstract Review System Scope 3](#_Toc500164528)

[1.4 Requirements and Assumptions 5](#_Toc500164529)

[1.5 E/R Diagram 7](#_Toc500164530)

[1.6 Relational Model 8](#_Toc500164531)

[1.6 Normalized Schema 9](#_Toc500164532)

[1.7 Normalized database schema 10](#_Toc500164533)

[1.8 Database Schema 13](#_Toc500164534)

[1.9 Database Record 20](#_Toc500164535)

[1.10 SQL Queries 23](#_Toc500164536)

[1.11 Views 24](#_Toc500164537)

[1.12 Procedures 25](#_Toc500164538)

[1.13 Reports 29](#_Toc500164539)

[1.14 Graphical User Interface 3](#_Toc500164540)2

# 1.3 Abstract Review System Scope

HENAAC conference planning, abstract reviews, poster competition, work with different

institutions. It need to be able to review folders that match with students and gives them

feedback about their work. The submissions must be in word. Edits can be made in word.

**Sequence of completion:**

Submission of work

Review by mentor

Feed back to the student

Edits or corrections to the work

Should be updated by student

Sent back for review

Approve

The abstract can be resubmitted only if it was not rejected. It should be able to notify

the reviewer that changes or corrections were made. It should show the deadline.

Students are going to be submitting their abstracts from more than 15 institutions.

Students are undergraduate and graduate from CS, EE, or Math. There are 2 levels of competition, graduate, and undergraduate.

It should store, last name, first name, major, classification, title of the abstract,

mentor/mentors, institution, keep track of mentor email, ethnicity, gender, and student email.

It also needs to be able to see how many abstracts have being summited, CAHSI colors,

and logo must be present. Students can submit, view, and update the abstract but they cannot

delete it. It must keep information from previous years. There are four types of users which are

Mentor, Review, Student, and Admin.

**Task to perform:**

The system must allow the reviewer to make changes and review the abstract.

Status of the system:

Approved – Ready for submission

Approved- with Corrections

Not Approved – Major Corrections (Can still submit)

Rejected (Cannot Submit again)

The system must allow the abstract to be time sensitive. Meaning that there is a time limit to upload the abstract. Posters are in editable form (PowerPoint, and Word)

Keep track of where the submission came (to make sure it’s a CHASI school)

Majors of the students

Names of participants

Classification of the students (Freshman, Sophomore, Junior, Senior)

Title of abstract

Who Is mentor(s) (Notify)

Mentor email

Ethnicity

Gender

Institution

Student email

# 1.4 Requirements and Assumptions

**Functional​ ​Requirements**

F1. The system shall only accept the abstract in a word file document.

F2. The system shall not allow a rejected abstract to be submitted again.

F3. The system shall not allow the students to delete of submissions.

F4. The system shall allow the mentors to view their student’s entries.

F5. The system shall record the administrator’s name, email, institution, and phone number.

F6. The system shall keep all reviews anonymous.

F7. The system shall keep record of students' names, emails, classifications, institutions, genders, ethnicities, and majors.

F8. The system shall keep record of mentors' and reviewers' name, email, gender, institution, title, and ethnicity.

F9. The system shall allow the administrator to set up deadlines for the reviewers and students

submissions.

F10. The system shall notify administrator of latest updates.

F11. The system shall allow the student to edit their submission and resubmit if their abstract is

not rejected.

F12. The system shall manage which mentors are supposed to review which submissions.

F13. The system shall generate reports for the administrator based on completion and status of

the abstracts.

F14. The system shall retain information from year to year to enable the administrator to contact the previous participants of new call for papers and other events.

F15. The system shall also house a review system for the posters in PowerPoint.

F16. The system shall display CAHSI logo and colors in the interface.

F17. The system shall provide separate interfaces for each type of user.

F18. The system shall be semi-reactive (able to resize).

F19. The system shall have color codes for the different statuses that an abstract may have.

**Functional Assumptions**

A1. The system will only accept posters in PowerPoint form. This will enable the program to conduct quick and easy correction mark up.

A2. The system shall save a copy of each entry and correction of the poster’s an abstract so that the administrator may, at her discretion see the progression of a submission.

A3. The system shall timestamp each submission and ensure that it is not submitted after the time it is due.

# 1.5 E/R Diagram

**A close up of a device

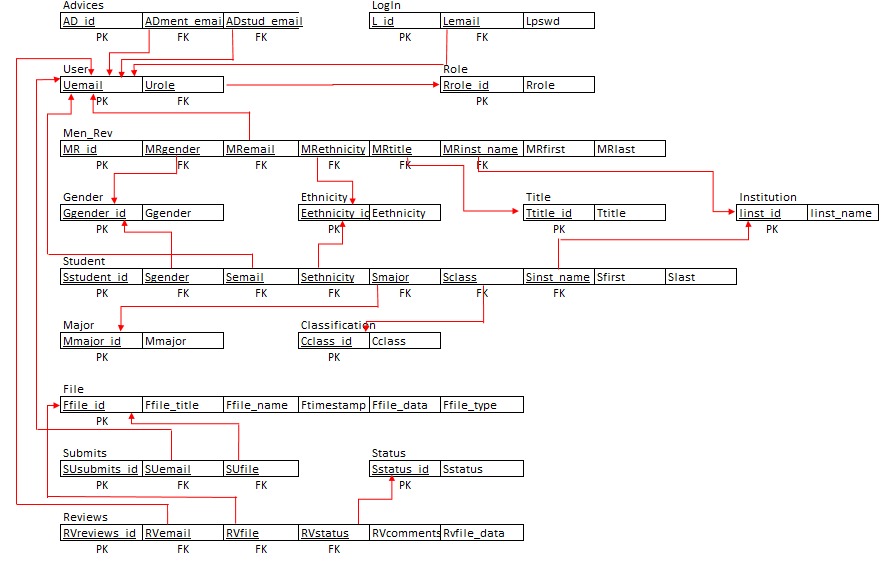
Description generated with high confidence**

# 1.6 Relational Model

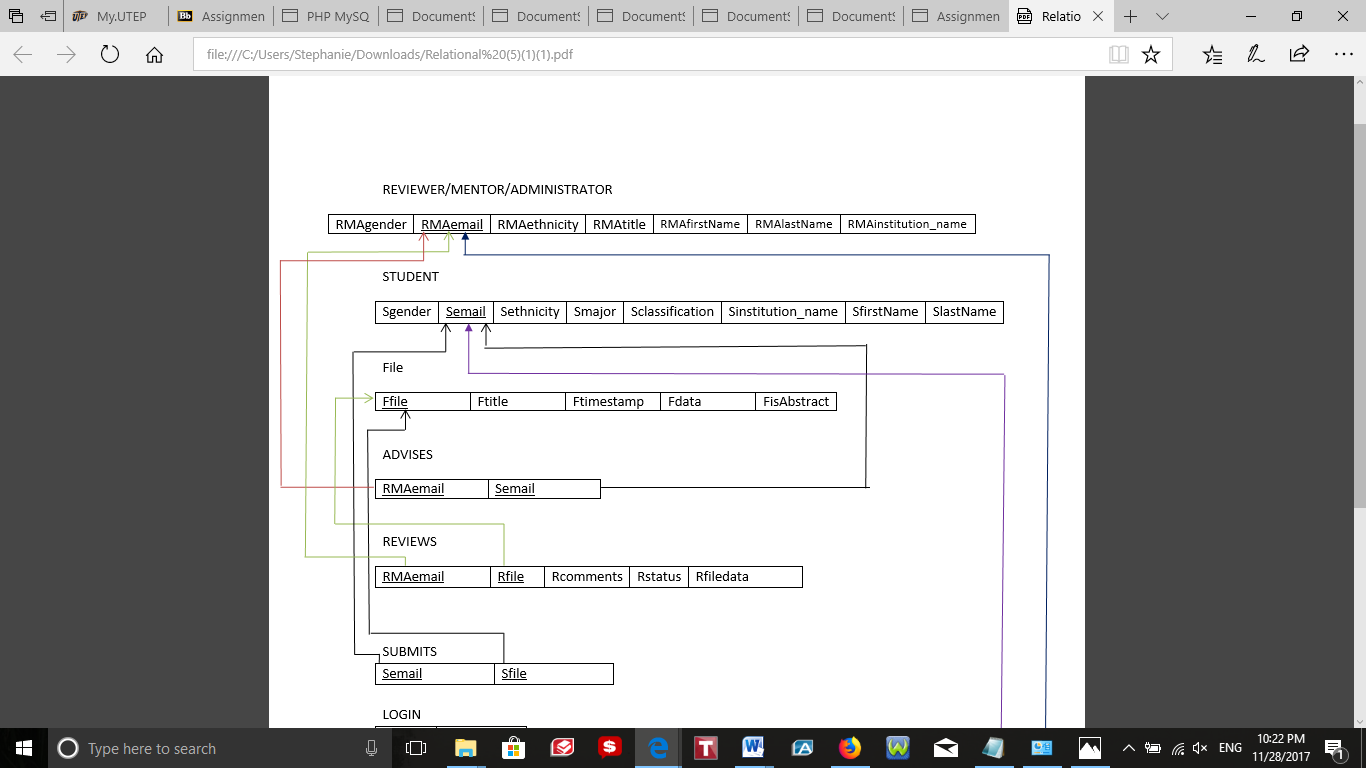
A screenshot of a cell phone

Description generated with very high confidence

# 1.6 Normalized Schema

****

# 1.7 Normalized database schema



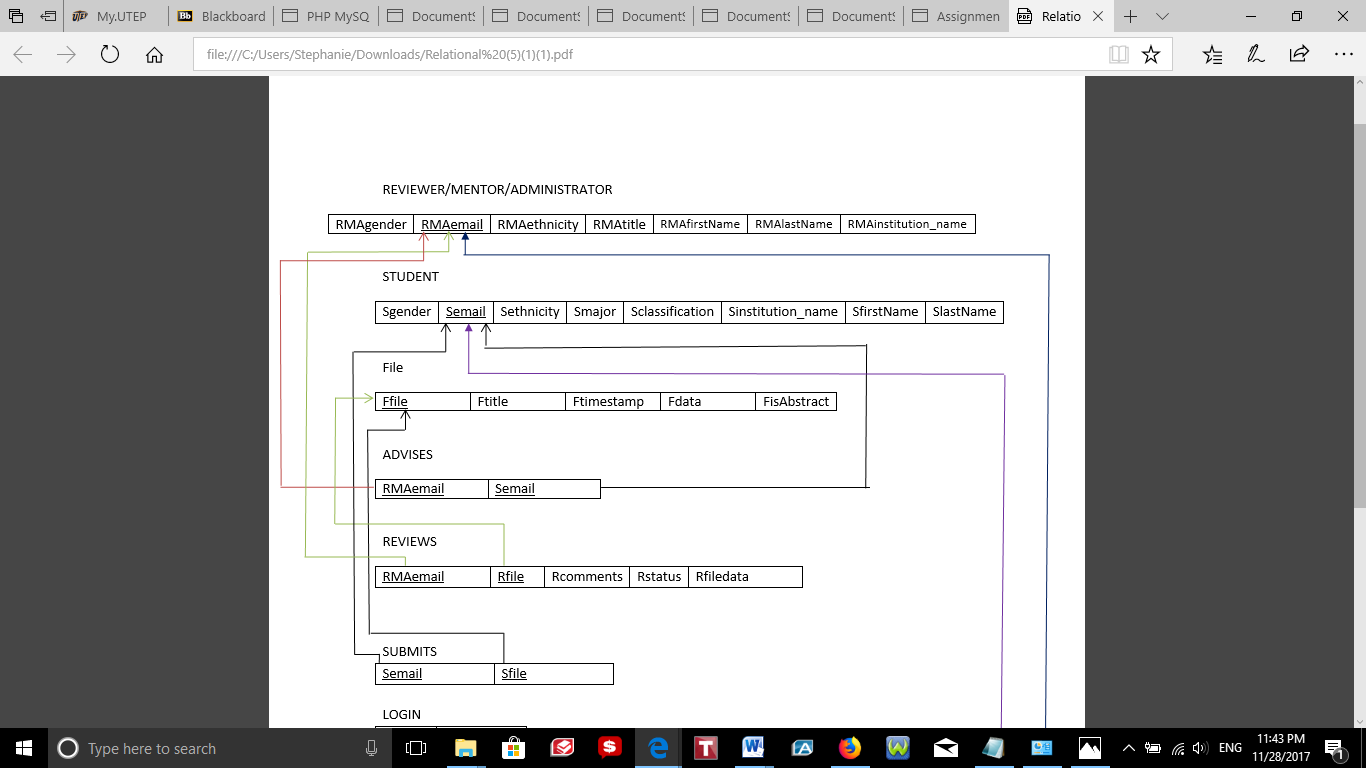
**Functional Dependencies**

RMAemail ->{RMAgender, RMAemail, RMAethnicity, RMAtitle, RMAfirstName, RMAlastname, RMAinstitution\_name}

**Description**

The primary key in this relation is RMAemail. The relation is in 1NF since all attributes contain atomic values. The 2NF is not violated, since every non-prime attributes are fully functionally dependent on the primary key, RMAemail. No non-prime attribute is transitively dependent on the primary key meaning that the 3NF is not violated.

Neither the attributes nor the functional dependency violates any normal forms. There is no need for corrections.



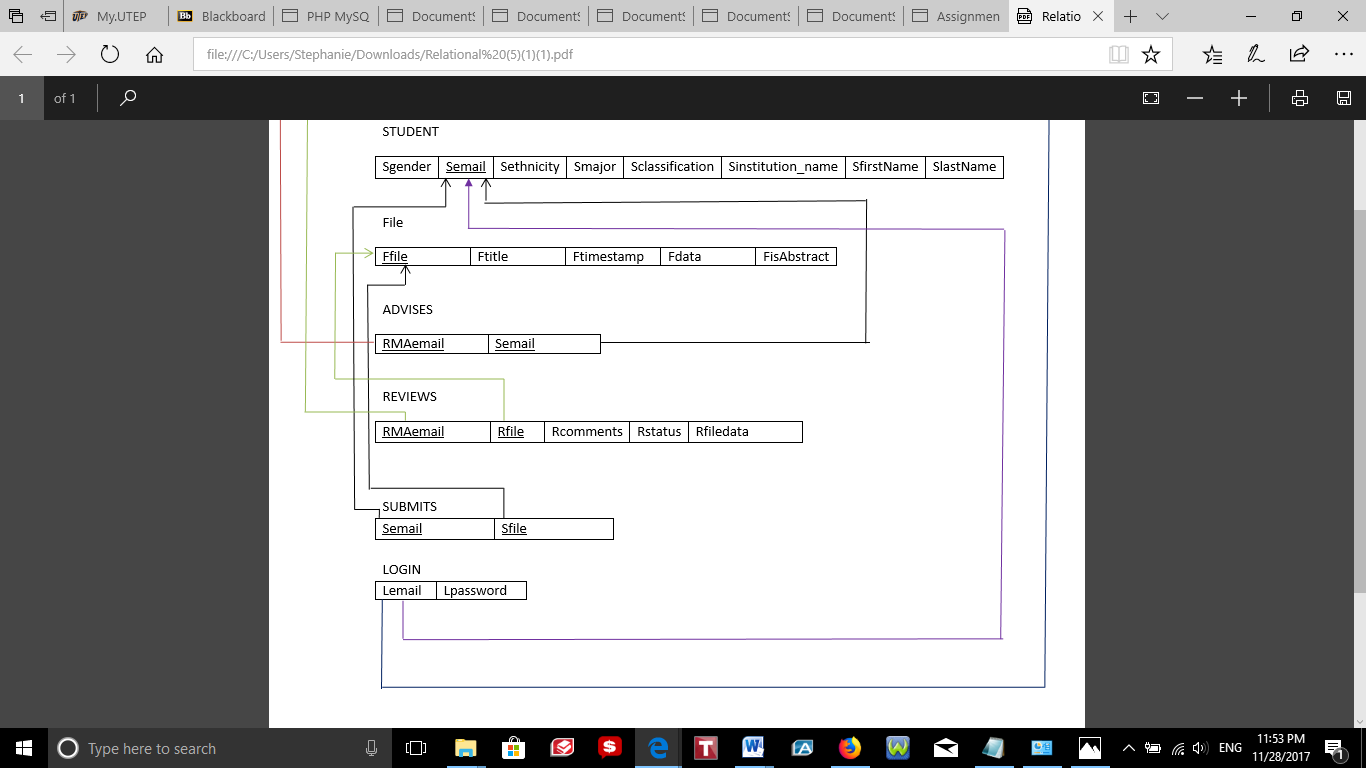
**Functional Dependencies**

Semail -> {Sgender, Semail, Sethnicity, Smajor, Sclassification, Sinstitution\_name, SfirstName, SlastName}

**Description**

Semail is the primary key for this relation. The relation is in 1NF since all attributes contain single value. The 2NF is not violated, since every non-prime attributes are fully functionally dependent on the primary key, Semail. No non-prime attribute is transitively dependent on the primary key meaning that the 3NF is not violated.

Neither the attributes nor the functional dependency violates any normal forms. There is no need for corrections.



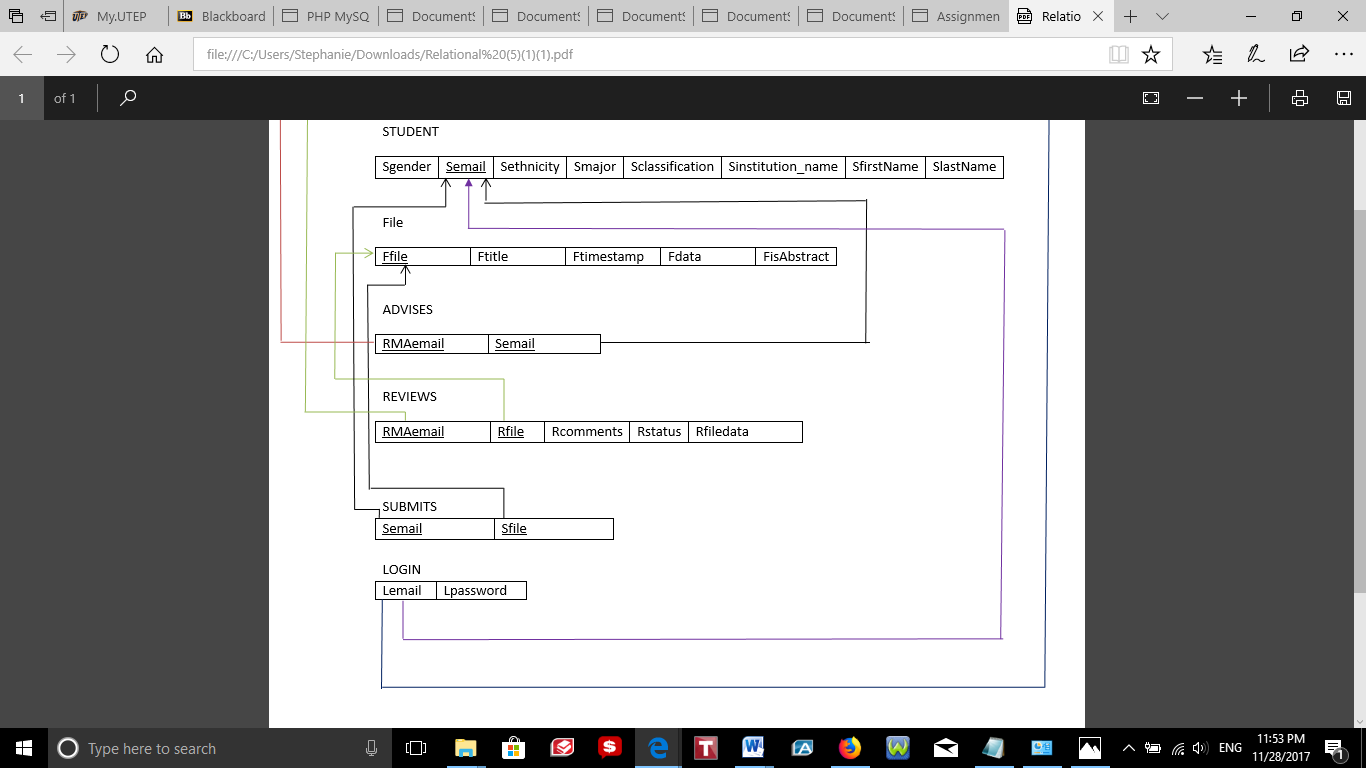
**Functional Dependencies**

Ffile ->{Ffile, Ftitle, Ftimestamp, Fdata, FisAbstract}

**Description**

The primary key in this relation is Ffile. The relation is in 1NF since all attributes contain atomic values. The 2NF is not violated, since every non-prime attributes are fully functionally dependent on the primary key, Ffile. No non-prime attribute is transitively dependent on the primary key meaning that the 3NF is not violated.

Neither the attributes nor the functional dependency violates any normal forms. There is no need for corrections.



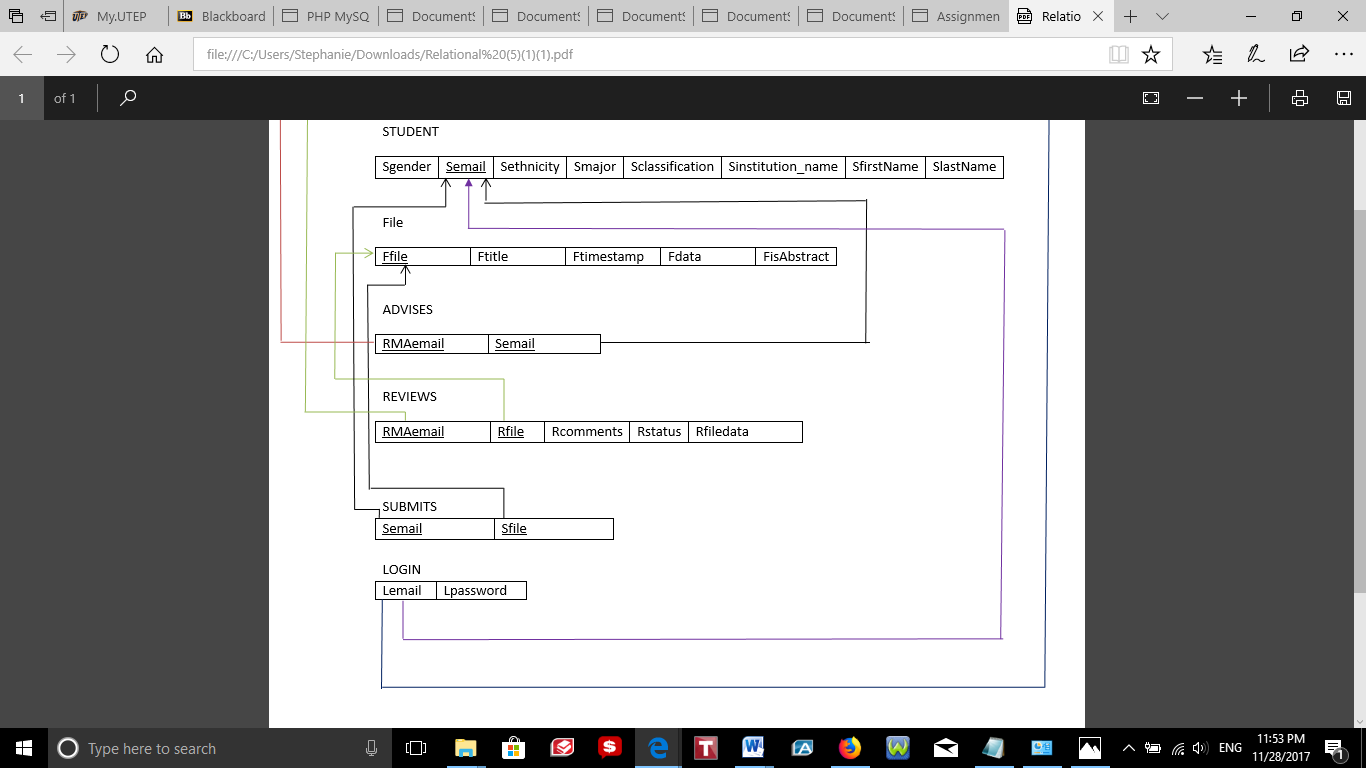
**Functional Dependencies**

RMAemail, Semail -> {RMAemail, Semail}

**Description**

Both RMAemail and Semail form the primary key. The relation is in 1NF since all attributes contain atomic values. The 2NF is not violated, since non-prime attributes are fully functionally dependent on the primary key (RMAemail and Semail). No non-prime attribute is transitively dependent on the primary key meaning that the 3NF is not violated.

Neither the attributes nor the functional dependency violates any normal forms. There is no need for corrections.



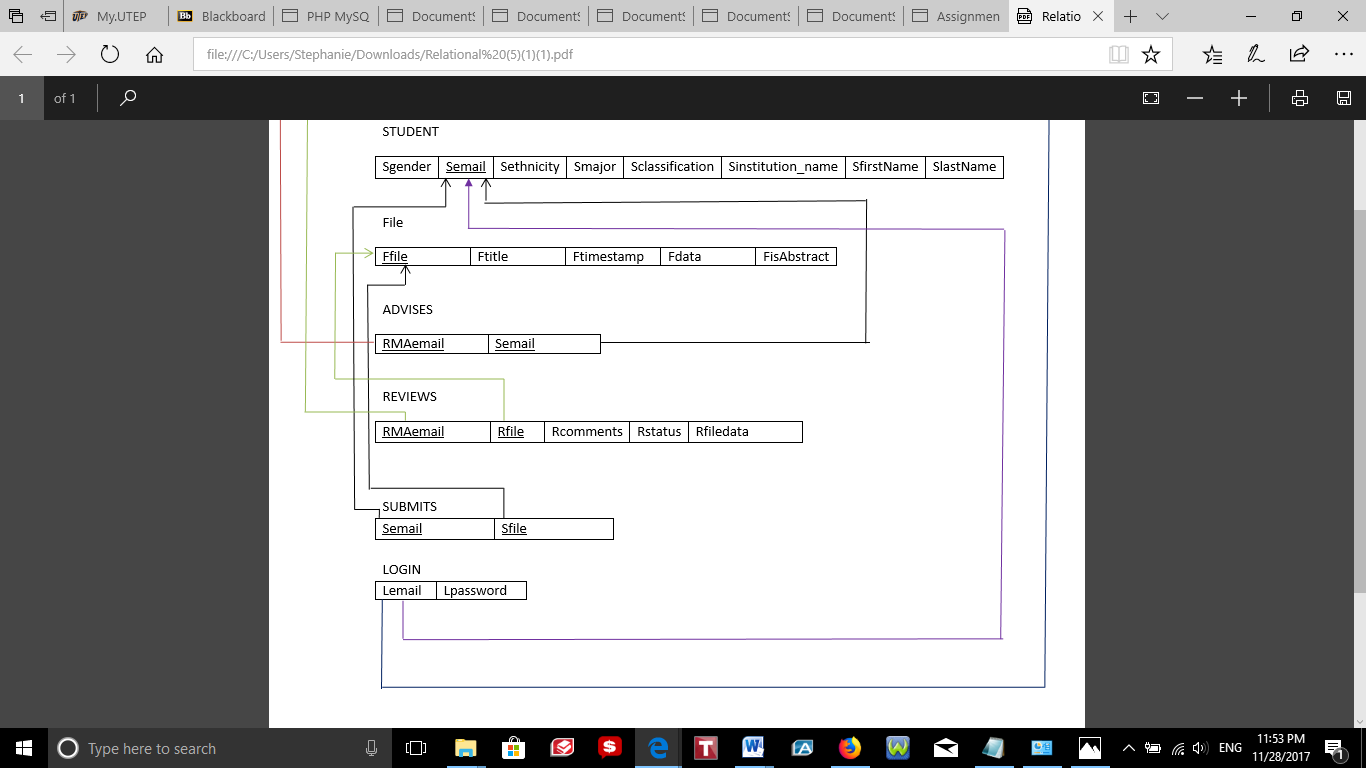
**Functional Dependencies**

RMAemail, Rfile -> {RMAemail, Rfile, Rcomments, Rstatus, Rfiledata}

**Description**

Both RMAemail and Rfile form the primary key. The relation is in 1NF since all attributes contain atomic values. The 2NF is not violated, since non-prime attributes are fully functionally dependent on the primary key (RMAemail and Rfile). No non-prime attribute is transitively dependent on the primary key meaning that the 3NF is not violated.

Neither the attributes nor the functional dependency violates any normal forms. There is no need for corrections.



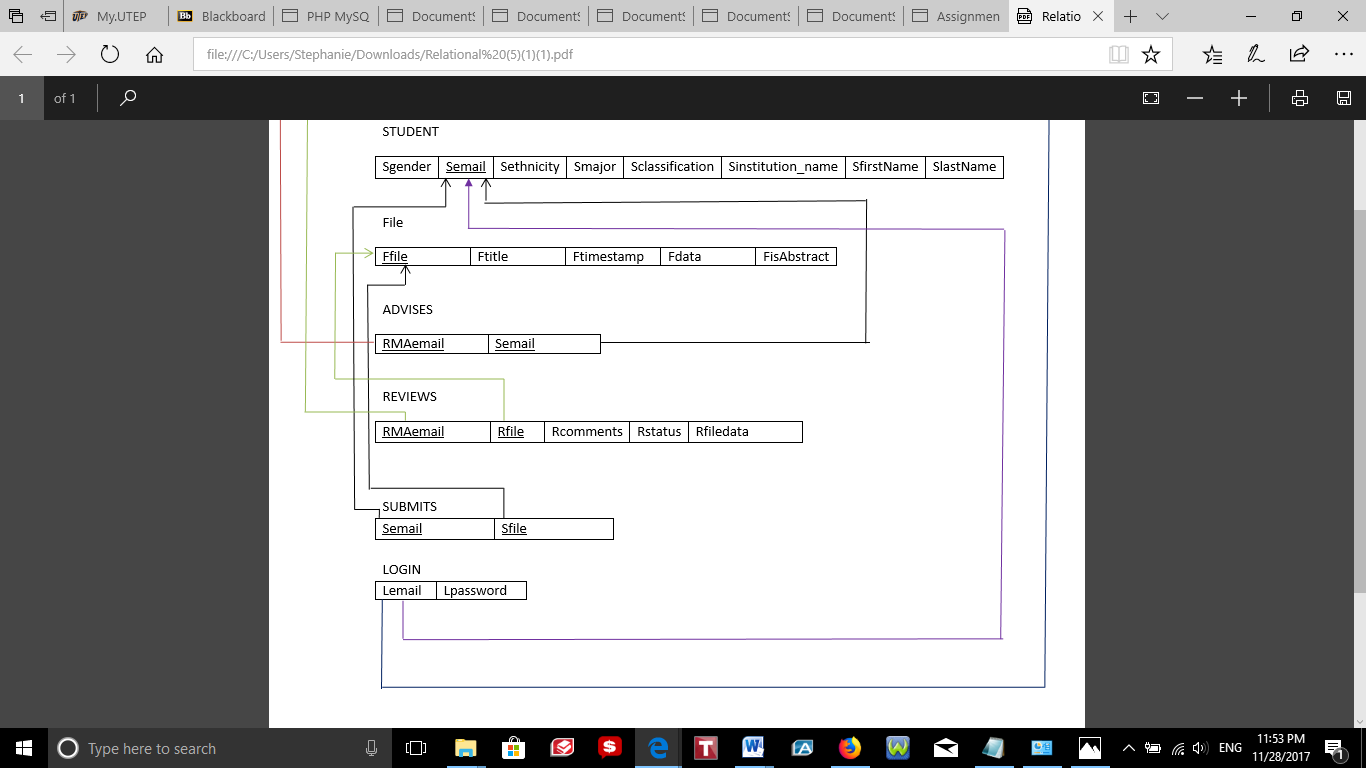
**Functional Dependencies**

Semail, Sfile -> {Semail, Sfile}

**Description**

Both Semail and Sfile form the primary key. The relation is in 1NF since all attributes contain atomic values. The 2NF is not violated, there are no non-prime attributes that depends partially on the primary key (Semail and Sfile). No non-prime attribute is transitively dependent on the primary key meaning that the 3NF is not violated.

Neither the attributes nor the functional dependency violates any normal forms. There is no need for corrections.



**Functional Dependencies**

Lemail -> {Lemail, Lpassword}

**Description**

Lemail is the primary key for this relation. The relation is in 1NF since all attributes contain single value. The 2NF is not violated, since the non-prime attribute (Lpassword) is fully functionally dependent on the primary key, Lemail. No non-key attribute is functionally determined by another non-key attribute.

Neither the attributes nor the functional dependency violates any normal forms. There is no need for corrections.

# 1.8 Database Schema

CREATE TABLE Gender

(

Ggender\_id INT(6) NOT NULL AUTO\_INCREMENT,

Ggender VARCHAR(15) NOT NULL,

PRIMARY KEY (Ggender\_id)

)ENGINE=InnoDB;

CREATE TABLE Ethnicity

(

Eethnicity\_id INT(6) NOT NULL AUTO\_INCREMENT,

Eethnicity VARCHAR(20) NOT NULL,

PRIMARY KEY (Eethnicity\_id)

)ENGINE=InnoDB;

CREATE TABLE Title

(

Ttitle\_id INT(6) NOT NULL AUTO\_INCREMENT,

Ttitle VARCHAR(50) NOT NULL,

PRIMARY KEY (Ttitle\_id)

)ENGINE=InnoDB;

CREATE TABLE Institution

(

Iinst\_id INT(6) NOT NULL AUTO\_INCREMENT,

Iinst\_name VARCHAR(50) NOT NULL,

PRIMARY KEY (Iinst\_id)

)ENGINE=InnoDB;

CREATE TABLE Major

(

Mmajor\_id INT(6) NOT NULL AUTO\_INCREMENT,

Mmajor VARCHAR(50) NOT NULL,

PRIMARY KEY (Mmajor\_id)

)ENGINE=InnoDB;

CREATE TABLE Role

(

Rrole\_id INT(6) NOT NULL AUTO\_INCREMENT,

Rrole VARCHAR(10) NOT NULL,

PRIMARY KEY (Rrole\_id)

)ENGINE=InnoDB;

CREATE TABLE Classification

(

Cclass\_id INT(6) NOT NULL AUTO\_INCREMENT,

Cclassification VARCHAR(50) NOT NULL,

PRIMARY KEY (Cclass\_id)

)ENGINE=InnoDB;

CREATE TABLE Status

(

Sstatus\_id INT(6) NOT NULL AUTO\_INCREMENT,

Sstatus VARCHAR(50) NOT NULL,

PRIMARY KEY (Sstatus\_id)

)ENGINE=InnoDB;

CREATE TABLE User

(

Uemail VARCHAR(50) NOT NULL,

Urole INT(6) NOT NULL,

PRIMARY KEY (Uemail),

FOREIGN KEY (Urole) REFERENCES Role(Rrole\_id)

)ENGINE=InnoDB;

CREATE TABLE File

(

Ffile\_id INT(6) NOT NULL AUTO\_INCREMENT,

Ffile\_title VARCHAR(20) NOT NULL,

Ffile\_name VARCHAR(50) NOT NULL,

Ftimestamp DATE NOT NULL,

Ffile\_data MEDIUMBLOB NOT NULL,

Ffile\_type BOOLEAN NOT NULL,

PRIMARY KEY (Ffile\_id)

)ENGINE=InnoDB;

CREATE TABLE Login

(

L\_id INT(6) NOT NULL AUTO\_INCREMENT,

Lemail VARCHAR(50) NOT NULL,

Lpswd VARCHAR(50) NOT NULL,

PRIMARY KEY (L\_id),

FOREIGN KEY (Lemail) REFERENCES User(Uemail)

ON DELETE CASCADE

ON UPDATE CASCADE

)ENGINE=InnoDB;

CREATE TABLE Advises

(

AD\_id INT(6) NOT NULL AUTO\_INCREMENT,

ADment\_email VARCHAR(50) NOT NULL,

ADstud\_email VARCHAR(50) NOT NULL,

PRIMARY KEY (AD\_id),

FOREIGN KEY (ADment\_email) REFERENCES User(Uemail)

ON DELETE CASCADE

ON UPDATE CASCADE,

FOREIGN KEY (ADstud\_email) REFERENCES User(Uemail)

ON DELETE CASCADE

ON UPDATE CASCADE

)ENGINE=InnoDB;

CREATE TABLE Submits

(

SUsubmits\_id INT(6) NOT NULL AUTO\_INCREMENT,

SUemail VARCHAR(50) NOT NULL,

SUfile\_id INT(6) NOT NULL,

PRIMARY KEY (SUsubmits\_id),

FOREIGN KEY (SUemail) REFERENCES User(Uemail)

ON DELETE CASCADE

ON UPDATE CASCADE,

FOREIGN KEY (SUfile\_id) REFERENCES File(Ffile\_id)

ON DELETE CASCADE

ON UPDATE CASCADE

)ENGINE=InnoDB;

CREATE TABLE Reviews

(

RVreviews\_id INT(6) NOT NULL AUTO\_INCREMENT,

RVemail VARCHAR(50) NOT NULL,

RVfile INT(6) NOT NULL,

RVstatus INT(6) NOT NULL,

RVcomments VARCHAR(100),

RVfile\_data MEDIUMBLOB NOT NULL,

PRIMARY KEY (RVreviews\_id),

FOREIGN KEY (RVemail) REFERENCES User(Uemail)

ON DELETE CASCADE

ON UPDATE CASCADE,

FOREIGN KEY (RVfile) REFERENCES File(Ffile\_id)

ON DELETE CASCADE

ON UPDATE CASCADE,

FOREIGN KEY (RVstatus) REFERENCES Status(Sstatus\_id)

ON DELETE CASCADE

ON UPDATE CASCADE

)ENGINE=InnoDB;

CREATE TABLE Men\_Rev

(

MR\_id INT(6) NOT NULL AUTO\_INCREMENT,

MRgender INT(6) NOT NULL,

MRemail VARCHAR(50) NOT NULL,

MRethnicity INT(6) NOT NULL,

MRtitle INT(6) NOT NULL,

MRinst\_name INT(6) NOT NULL,

MRfirst VARCHAR(50) NOT NULL,

MRlast VARCHAR(50) NOT NULL,

PRIMARY KEY (MR\_id),

FOREIGN KEY (MRgender) REFERENCES Gender(Ggender\_id)

ON DELETE CASCADE

ON UPDATE CASCADE,

FOREIGN KEY (MRemail) REFERENCES User(Uemail)

ON DELETE CASCADE

ON UPDATE CASCADE,

FOREIGN KEY (MRethnicity) REFERENCES Ethnicity(Eethnicity\_id)

ON DELETE CASCADE

ON UPDATE CASCADE,

FOREIGN KEY (MRtitle) REFERENCES Title(Ttitle\_id)

ON DELETE CASCADE

ON UPDATE CASCADE,

FOREIGN KEY (MRinst\_name) REFERENCES Institution(Iinst\_id)

ON DELETE CASCADE

ON UPDATE CASCADE

)ENGINE=InnoDB;

CREATE TABLE Student

(

Sstudent\_id INT(6) NOT NULL AUTO\_INCREMENT,

Sgender INT(6) NOT NULL,

Semail VARCHAR(50) NOT NULL,

Sethnicity INT(6) NOT NULL,

Smajor INT(6) NOT NULL,

Sclass INT(6) NOT NULL,

Sinst\_name INT(6) NOT NULL,

Sfirst VARCHAR(50) NOT NULL,

Slast VARCHAR(50) NOT NULL,

PRIMARY KEY (Sstudent\_id),

FOREIGN KEY (Sgender) REFERENCES Gender(Ggender\_id)

ON DELETE CASCADE

ON UPDATE CASCADE,

FOREIGN KEY (Semail) REFERENCES User(Uemail)

ON DELETE CASCADE

ON UPDATE CASCADE,

FOREIGN KEY (Sethnicity) REFERENCES Ethnicity(Eethnicity\_id)

ON DELETE CASCADE

ON UPDATE CASCADE,

FOREIGN KEY (Smajor) REFERENCES Major(Mmajor\_id)

ON DELETE CASCADE

ON UPDATE CASCADE,

FOREIGN KEY (Sclass) REFERENCES Classification(Cclass\_id)

ON DELETE CASCADE

ON UPDATE CASCADE,

FOREIGN KEY (Sinst\_name) REFERENCES Institution(Iinst\_id)

ON DELETE CASCADE

ON UPDATE CASCADE

)ENGINE=InnoDB;

# 1.9 Database Record

All of the inserted Records follow this pattern in order to attain the correct look up reference numbers. This is an example provided from the student sign up page.

if($\_SERVER["REQUEST\_METHOD"] == "POST") {

$username = mysqli\_real\_escape\_string($db,$\_POST['username']);

$password = mysqli\_real\_escape\_string($db,$\_POST['password']);

$last = mysqli\_real\_escape\_string($db,$\_POST['lname']);

$first = mysqli\_real\_escape\_string($db,$\_POST['fname']);

$gender = mysqli\_real\_escape\_string($db,$\_POST['gender']);

$institution = mysqli\_real\_escape\_string($db,$\_POST['institution']);

$ethnicity = mysqli\_real\_escape\_string($db,$\_POST['ethnicity']);

$major = mysqli\_real\_escape\_string($db,$\_POST['major']);

$class = mysqli\_real\_escape\_string($db,$\_POST['classification']);

//Checks if the entry exists in the lookup table

$checkGender = "SELECT \* FROM Gender WHERE Ggender = '$gender'";

$checkGenderResult = mysqli\_query($db,$checkGender);

if($checkGenderResult){

$checkGenderResultRow = mysqli\_fetch\_array($checkGenderResult,MYSQLI\_ASSOC);

//Gets the lookup value to insert into the main table

$genderLookupNum = $checkGenderResultRow['Ggender\_id'];

}

//If it doesnt exist, the entry is added to the table and its lookup value is found.

else{

$insertGender = "INSERT INTO Gender (Ggender) VALUES ('$gender')";

mysqli\_query($db,$insertGender);

$checkGender = "SELECT \* FROM Gender WHERE Ggender = '$gender'";

$checkGenderResult = mysqli\_query($db,$checkGender);

$checkGenderResultRow = mysqli\_fetch\_array($checkGenderResult,MYSQLI\_ASSOC);

$genderLookupNum = $checkGenderResultRow['Ggender\_id'];

}

//Similar process for the rest of the variables

$checkInstituiton = "SELECT \* FROM Instituiton WHERE Iinst\_name = '$institution'";

$checkInstituitonResult = mysqli\_query($db,$checkInstituiton);

if($checkInstituitonResult){

$checkInstituitonResultRow = mysqli\_fetch\_array($checkInstituitonResult,MYSQLI\_ASSOC);

$InstitutionLookupNum = $checkInstituitonResultRow['Iinst\_id'];

}

else{

$insertInstitution = "INSERT INTO Institution (Iinst\_name) VALUES ('$institution')";

mysqli\_query($db,$insertInstitution);

$checkInstitution = "SELECT \* FROM Institution WHERE Iinst\_name = '$institution'";

$checkInstitutionResult = mysqli\_query($db,$checkInstitution);

$checkInstitutionResultRow = mysqli\_fetch\_array($checkInstitutionResult,MYSQLI\_ASSOC);

$InstitutionLookupNum = $checkInstitutionResultRow['Iinst\_id'];

}

$checkMajor = "SELECT \* FROM Major WHERE Mmajor = '$major'";

$checkMajorResult = mysqli\_query($db,$checkMajor);

if($checkMajorResult){

$checkMajorResultRow = mysqli\_fetch\_array($checkMajorResult,MYSQLI\_ASSOC);

$majorLookupNum = $checkMajorResultRow['Mmajor\_id'];

}

else{

$insertMajor = "INSERT INTO Major (Mmajor) VALUES ('$major')";

mysqli\_query($db,$insertMajor);

$checkMajor = "SELECT \* FROM Major WHERE Mmajor = '$major'";

$checkMajorResult = mysqli\_query($db,$checkMajor);

$checkMajorResultRow = mysqli\_fetch\_array($checkMajorResult,MYSQLI\_ASSOC);

$majorLookupNum = $checkMajorResultRow['Mmajor\_id'];

}

$checkEthnicity = "SELECT \* FROM Ethnicity WHERE Eethnicity = '$ethnicity'";

$checkEthnicityResult = mysqli\_query($db,$checkEthnicity);

if($checkEthnicityResult){

$checkEthnicityResultRow = mysqli\_fetch\_array($checkEthnicityResult,MYSQLI\_ASSOC);

$ethnicityLookupNum = $checkEthnicityResultRow['Eethnicity\_id'];

}

else{

$insertEthnicity = "INSERT INTO Ethnicity (Eethnicity) VALUES ('$ethnicity')";

mysqli\_query($db,$insertEthnicity);

$checkEthnicity = "SELECT \* FROM Ethnicity WHERE Eethnicity = '$ethnicity'";

$checkEthnicityResult = mysqli\_query($db,$checkEthnicity);

$checkEthnicityResultRow = mysqli\_fetch\_array($checkEthnicityResult,MYSQLI\_ASSOC);

$ethnicityLookupNum = $checkEthnicityResult['Eethnicity\_id'];

}

$checkClassification = "SELECT \* FROM Classification WHERE Cclassification = '$class'";

$checkClassificationResult = mysqli\_query($db,$checkClassification);

if($checkClassificationResult){

$checkClassificationResultRow = mysqli\_fetch\_array($checkClassificationResult,MYSQLI\_ASSOC);

$classLookupNum = $checkClassificationResultRow['Cclass\_id'];

}

else{

$insertClassification = "INSERT INTO Classification (Cclassification) VALUES ('$class')";

$checkDis = mysqli\_query($db,$insertClassification);

$checkClassification = "SELECT \* FROM Classification WHERE Cclassification = '$class'";

$checkClassificationResult = mysqli\_query($db,$checkClassification);

$checkClassificationResultRow = mysqli\_fetch\_array($checkClassificationResult,MYSQLI\_ASSOC);

$classLookupNum = $checkClassificationResult['Cclass\_id'];

}

$checkRole = "SELECT \* FROM Role WHERE Rrole = 'student'";

$checkRoleResult = mysqli\_query($db,$checkRole);

if($checkRoleResult){

$checkRoleResultRow = mysqli\_fetch\_array($checkRoleResult,MYSQLI\_ASSOC);

$studentLookupNum = $checkRoleResultRow['Rrole\_id'];

}

else{

$insertRole = "INSERT INTO Role (Rrole) VALUES ('student')";

mysqli\_query($db,$insertRole);

$checkRole = "SELECT \* FROM Role WHERE Rrole = 'student'";

$checkRoleResult = mysqli\_query($db,$checkRole);

$checkRoleResultRow = mysqli\_fetch\_array($checkRoleResult,MYSQLI\_ASSOC);

$studentLookupNum = $checkRoleResult['Rrole\_id'];

}

//Insert User

$insertUser = "INSERT INTO User (Uemail, Urole) VALUES ('$username', '$studentLookupNum')";

mysqli\_query($db,$insertUser);

//Building the main table entry with the lookup values found.

$insertStudent = "INSERT INTO Student (Sgender, Semail, Sethnicity, Smajor, Sclass, Sinst\_name, Sfirst, Slast) VALUES ('$genderLookupNum','$username','$ethnicityLookupNum','$majorLookupNum','$classLookupNum','$InstitutionLookupNum','$first','$last')";

mysqli\_query($db,$insertStudent);

header("Location: signUpSuccessful.php");

# 1.10 SQL Queries

*Satisfies Requirement 7:*

This output shows the students’ first name, last email, classifications, institutions, genders, ethnicities, and majors.

SELECT U.Uemail, S.Sfirst, S.Slast, G.Ggender, E.Eethnicity, M.Mmajor, C.Cclassification, I.Iinst\_name

FROM Student S

JOIN User U on U.Uemail = S.Semail

JOIN Gender G on G.Ggender\_id = S.Sgender

JOIN Ethnicity E on E.Eethnicity\_id = S.Sethnicity

JOIN Major M on M.Mmajor\_id = S.Smajor

JOIN Classification C on C.Cclass\_id = S.Sclass

JOIN Institution I on I.Iinst\_id = S.Sinst\_name;

*Satisfies Requirement 8:*

This output shows the mentors’ and reviewers' name, email, gender, institution, title, and ethnicity.

SELECT U.Uemail , R.MRfirst, R.MRlast, G.Ggender, E.Eethnicity, T.Ttitle, I.Iinst\_name

FROM Men\_Rev R

JOIN User U on U.Uemail = R.MRemail

JOIN Gender G on G.Ggender\_id = R.MRgender

JOIN Ethnicity E on E.Eethnicity\_id = R.MRethnicity

JOIN Title T on T.Ttitle\_id = R.MRtitle

JOIN Institution I on I.Iinst\_id= R.MRinst\_name;

# 1.11 Views

For the views, we had some particularly complicated queries in order to look at the students and mentors/reviewers. Aside from the general reports, we also had a filtering section that utilized these views to speed up the filtering process.

CREATE VIEW StudentRoster (Student\_Email, First\_name, Last\_name, S\_Gender, S\_Ethnicity, S\_Major, S\_Classification, S\_Institution) AS

SELECT U.Uemail, S.Sfirst, S.Slast, G.Ggender, E.Eethnicity, M.Mmajor, C.Cclassification, I.Iinst\_name

FROM Student S

JOIN User U on U.Uemail = S.Semail

JOIN Gender G on G.Ggender\_id = S.Sgender

JOIN Ethnicity E on E.Eethnicity\_id = S.Sethnicity

JOIN Major M on M.Mmajor\_id = S.Smajor

JOIN Classification C on C.Cclass\_id = S.Sclass

JOIN Institution I on I.Iinst\_id = S.Sinst\_name

WHERE I.Iinst\_id= 1;

CREATE VIEW MentorRoster (Men\_Rev\_email, M\_R\_First\_Name, M\_R\_Last\_Name, M\_R\_Gender, M\_R\_Ethnicity, M\_R\_Title, M\_R\_Institution) AS

SELECT U.Uemail , R.MRfirst, R.MRlast, G.Ggender, E.Eethnicity, T.Ttitle, I.Iinst\_name

FROM Men\_Rev R

JOIN User U on U.Uemail = R.MRemail

JOIN Gender G on G.Ggender\_id = R.MRgender

JOIN Ethnicity E on E.Eethnicity\_id = R.MRethnicity

JOIN Title T on T.Ttitle\_id = R.MRtitle

JOIN Institution I on I.Iinst\_id= R.MRinst\_name

WHERE I.Iinst\_id = 2;

# 1.12 Procedures

**Procedure students’ gender**

\d//

CREATE PROCEDURE GenderStudentTotal(IN sgender INT(6))

BEGIN

SELECT count(\*) AS totalGender

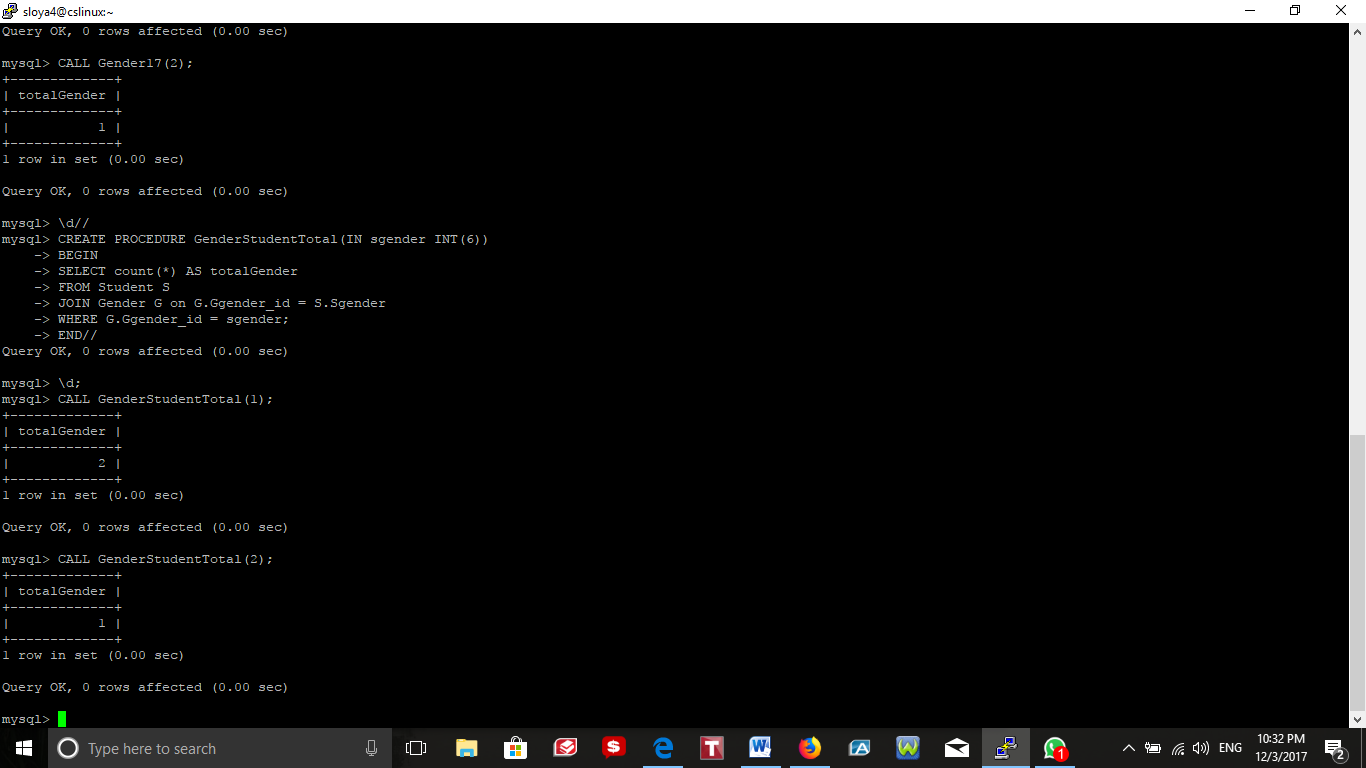
FROM Student S

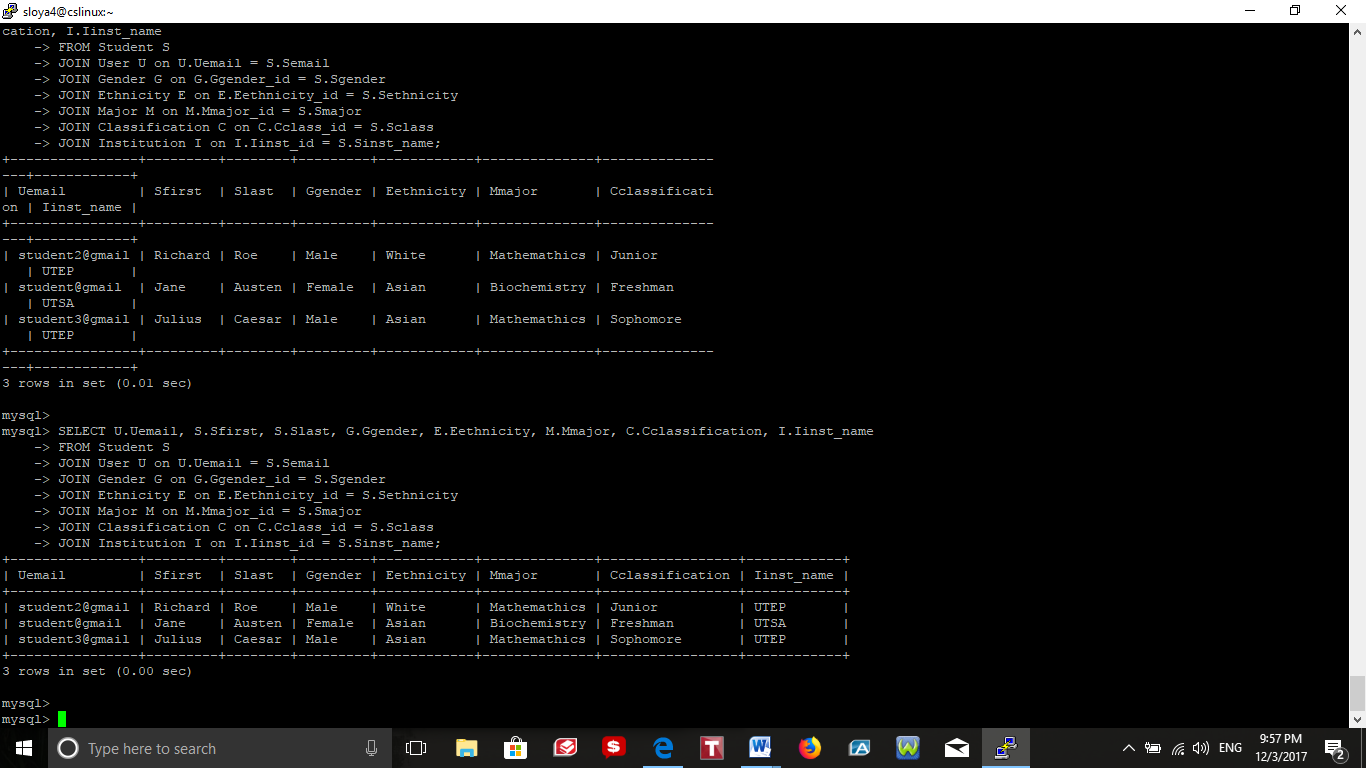
JOIN Gender G on G.Ggender\_id = S.Sgender

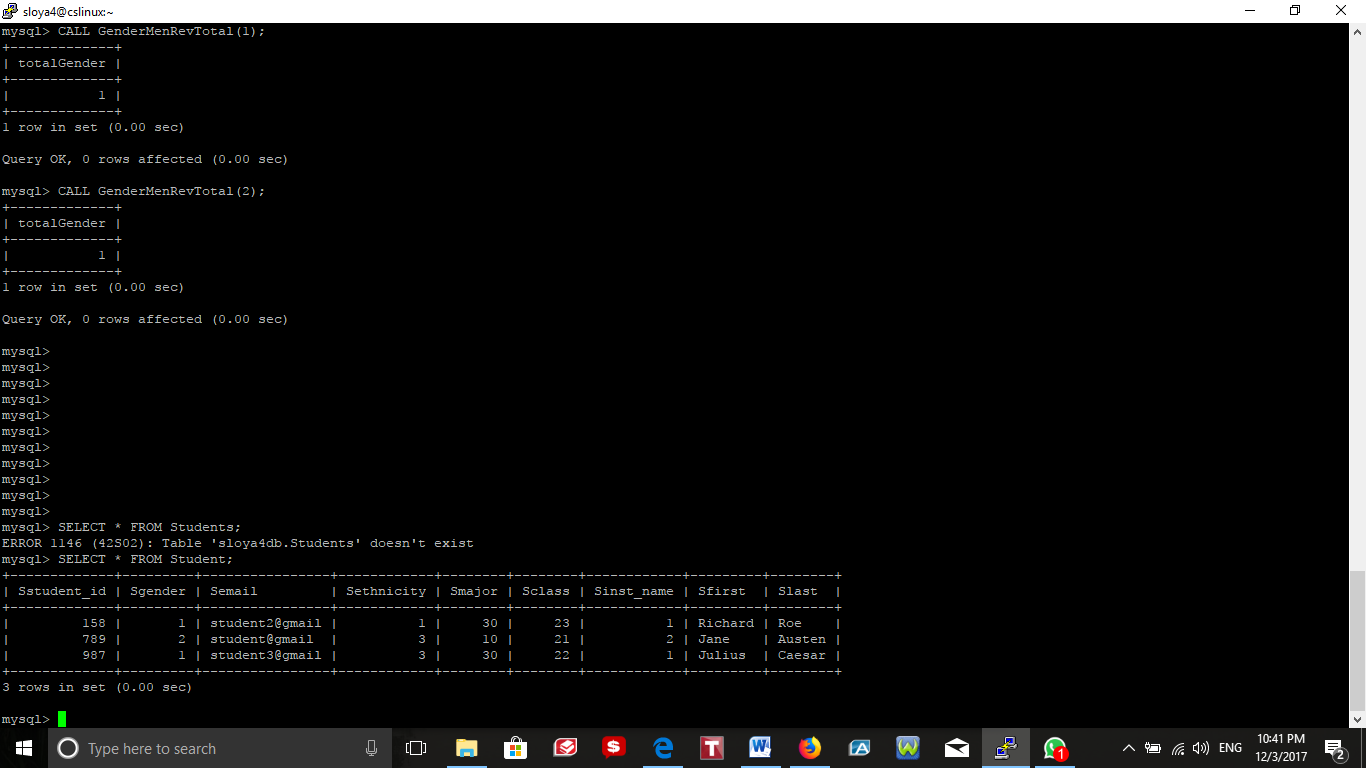
WHERE G.Ggender\_id = sgender;

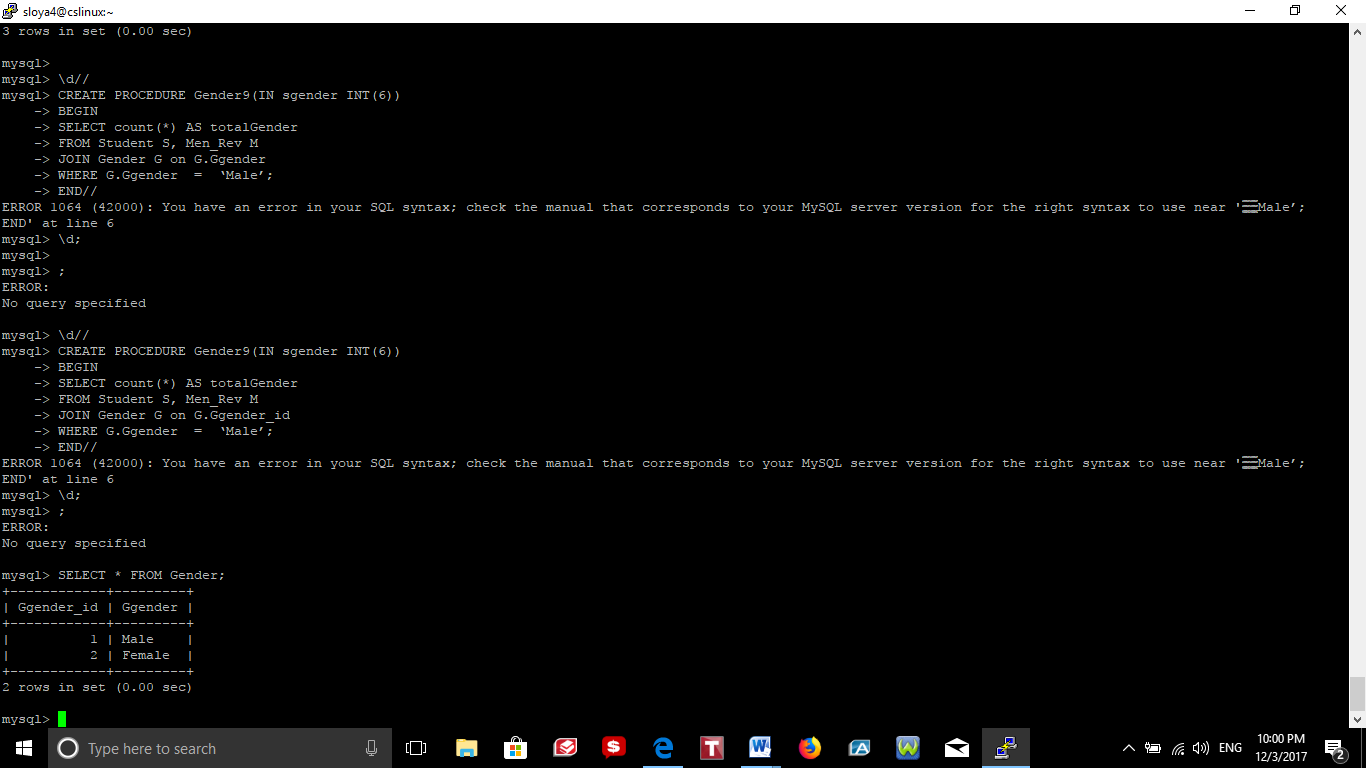
END//

\d;









**Procedure students’ Mentor/Reviewer**

\d//

CREATE PROCEDURE GenderMenRevTotal(IN sgender INT(6))

BEGIN

SELECT count(\*) AS totalGender

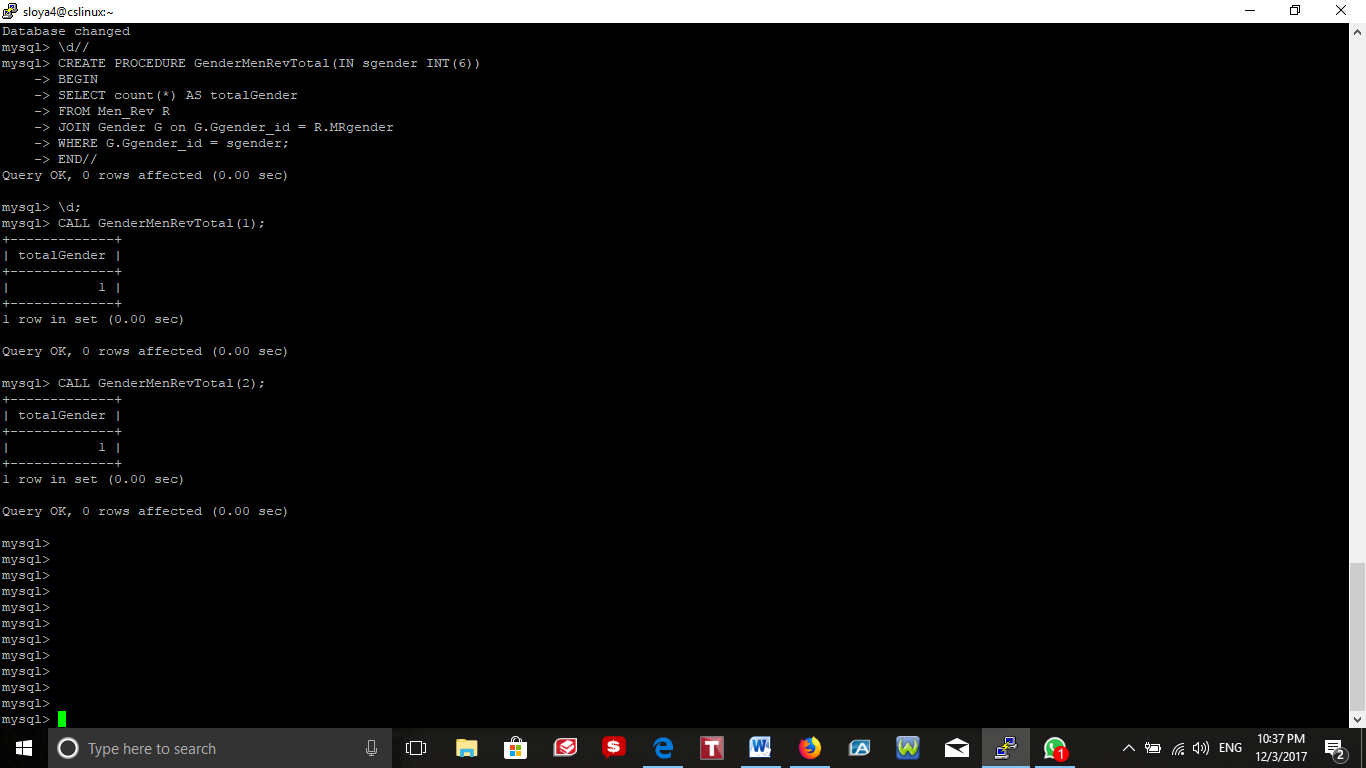
FROM Men\_Rev R

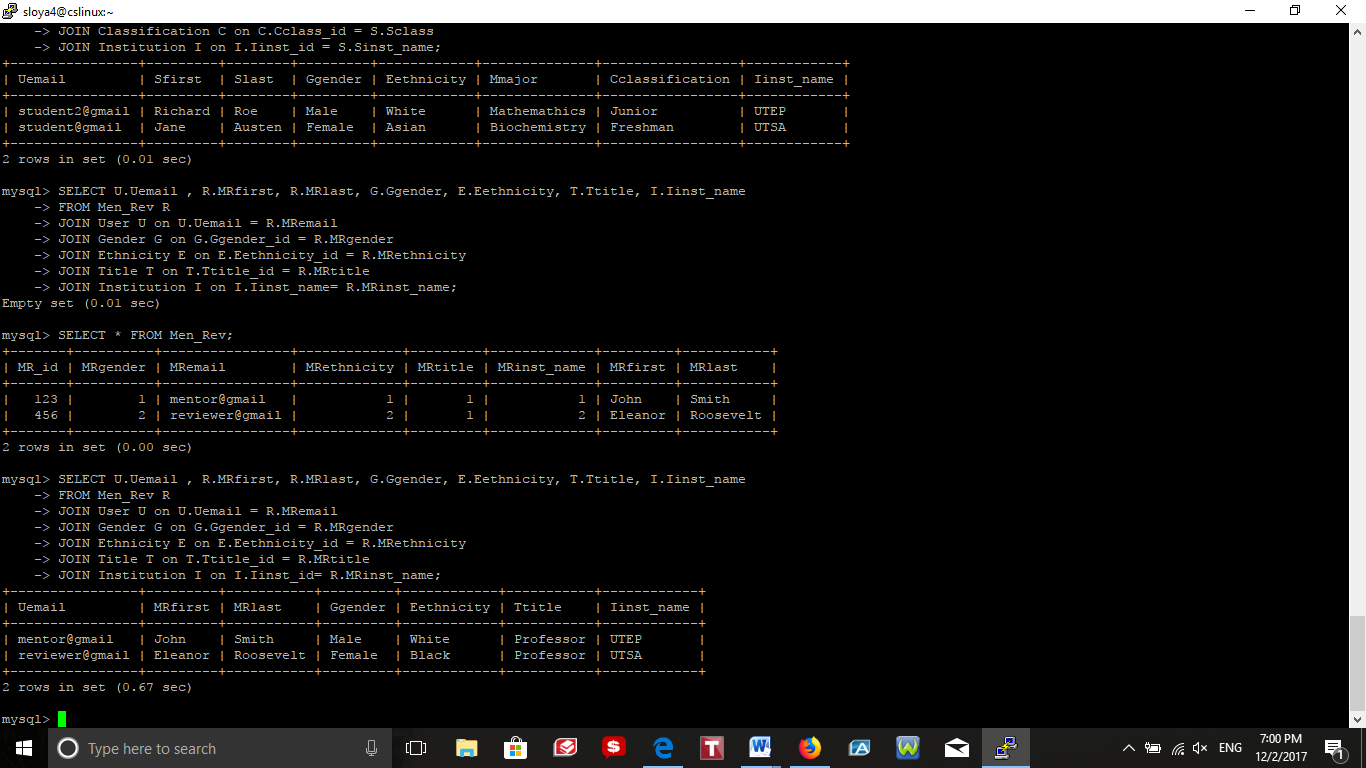
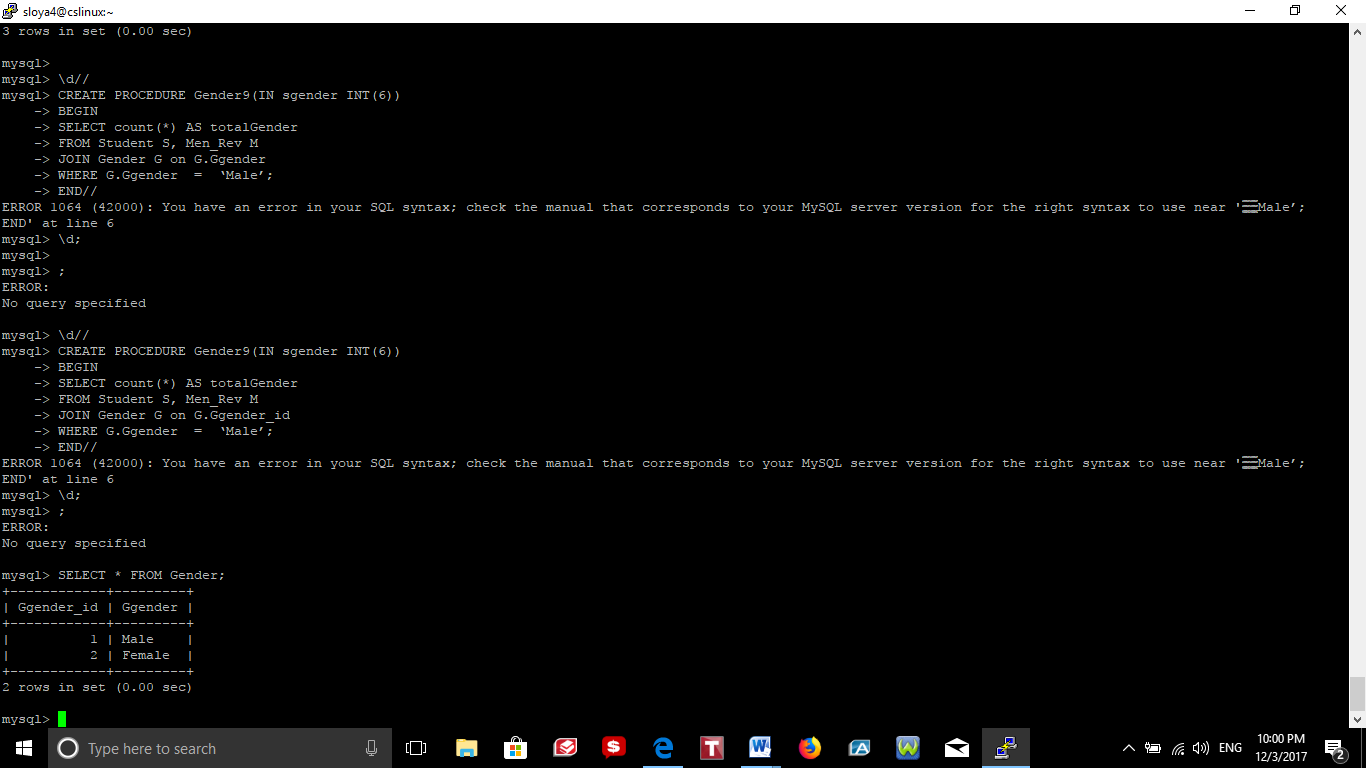
JOIN Gender G on G.Ggender\_id = R.MRgender

WHERE G.Ggender\_id = sgender;

END//

\d;





# 1.13 Reports

*Total number of all the students*

SELECT COUNT(\*) AS Total\_Students

FROM Student ;

*Number of students per institution*

SELECT I.Iinst\_name, COUNT(\*) AS Total

FROM Student S

JOIN Institution I on I.Iinst\_id= S.Sinst\_name

WHERE I.Iinst\_id= S.Sinst\_name

GROUP BY I.Iinst\_id;

*Number of students per gender*

SELECT G.Ggender, COUNT(\*) AS Total

FROM Student S

JOIN Gender G on G.Ggender\_id = S.Sgender

WHERE G.Ggender\_id = S.Sgender

GROUP BY G.Ggender\_id;

*Number of students per ethnicity*

SELECT E.Eethnicity, COUNT(\*) AS Total

FROM Student S

JOIN Ethnicity E on E.Eethnicity\_id = S.Sethnicity

WHERE E.Eethnicity\_id = S.Sethnicity

GROUP BY E.Eethnicity\_id;

*Number of students per major*

SELECT M.Mmajor, COUNT(\*) AS Total

FROM Student S

JOIN Major M on M.Mmajor\_id = S.Smajor

WHERE M.Mmajor\_id = S.Smajor

GROUP BY M.Mmajor\_id;

*Number of students per classification*

SELECT C.Cclassification, COUNT(\*) AS Total

FROM Student S

JOIN Classification C on C.Cclass\_id = S.Sclass

WHERE C.Cclass\_id = S.Sclass

GROUP BY C.Cclass\_id;

*Total number of all the Mentors/Reviewers*

SELECT COUNT(\*) AS Total\_Mentors\_Reviewers

FROM Men\_Rev;

*Number of reviewer /mentor per institution*

SELECT I.Iinst\_name, COUNT(\*) AS Total

FROM Men\_Rev R

JOIN Institution I on I.Iinst\_id= R.MRinst\_name

WHERE I.Iinst\_id= R.MRinst\_name

GROUP BY I.Iinst\_id;

*Number of reviewer /mentor per gender*

SELECT G.Ggender, COUNT(\*) AS Total

FROM Men\_Rev R

JOIN Gender G on G.Ggender\_id= R.MRgender

WHERE G.Ggender\_id= R.MRgender

GROUP BY G.Ggender\_id;

*Number of reviewer /mentor per ethnicity*

SELECT E.Eethnicity, COUNT(\*) AS Total

FROM Men\_Rev R

JOIN Ethnicity E on E.Eethnicity\_id = R.MRethnicity

WHERE E.Eethnicity\_id = R.MRethnicity

GROUP BY E.Eethnicity\_id;

*Number of reviewer /mentor per title*

SELECT T.Ttitle, COUNT(\*) AS Total

FROM Men\_Rev R

JOIN Title T on T.Ttitle\_id = R.MRtitle

WHERE T.Ttitle\_id = R.MRtitle

GROUP BY T.Ttitle\_id;

# 1.14 Graphical User Interface

