**My Better Concordia**

**DLC353\_2**

Christiano Bianchet (2703953)

Anuvret Bhatotia (40014439)

Tawfiq Jawhar (26535186)

Andrei Chira (9680039)

**Summary**

*My Better Concordia* is a web application intended for professors to keep track of their roles as teachers, researchers, grant receivers, committee members and editorial board members. Our system uses a MySQL server as as database containing all the information regarding the teachers. The teacher is able to login using his/her credentials (email and password) and view a list of all courses they have taught. They can then view course information regarding these courses as well as the profiles of all the students they have taught. Grades of these students can be viewed (final letter grades, course component grades) as well as the average grade in a specific course. All of this information can also be filtered by semester or time-frame.

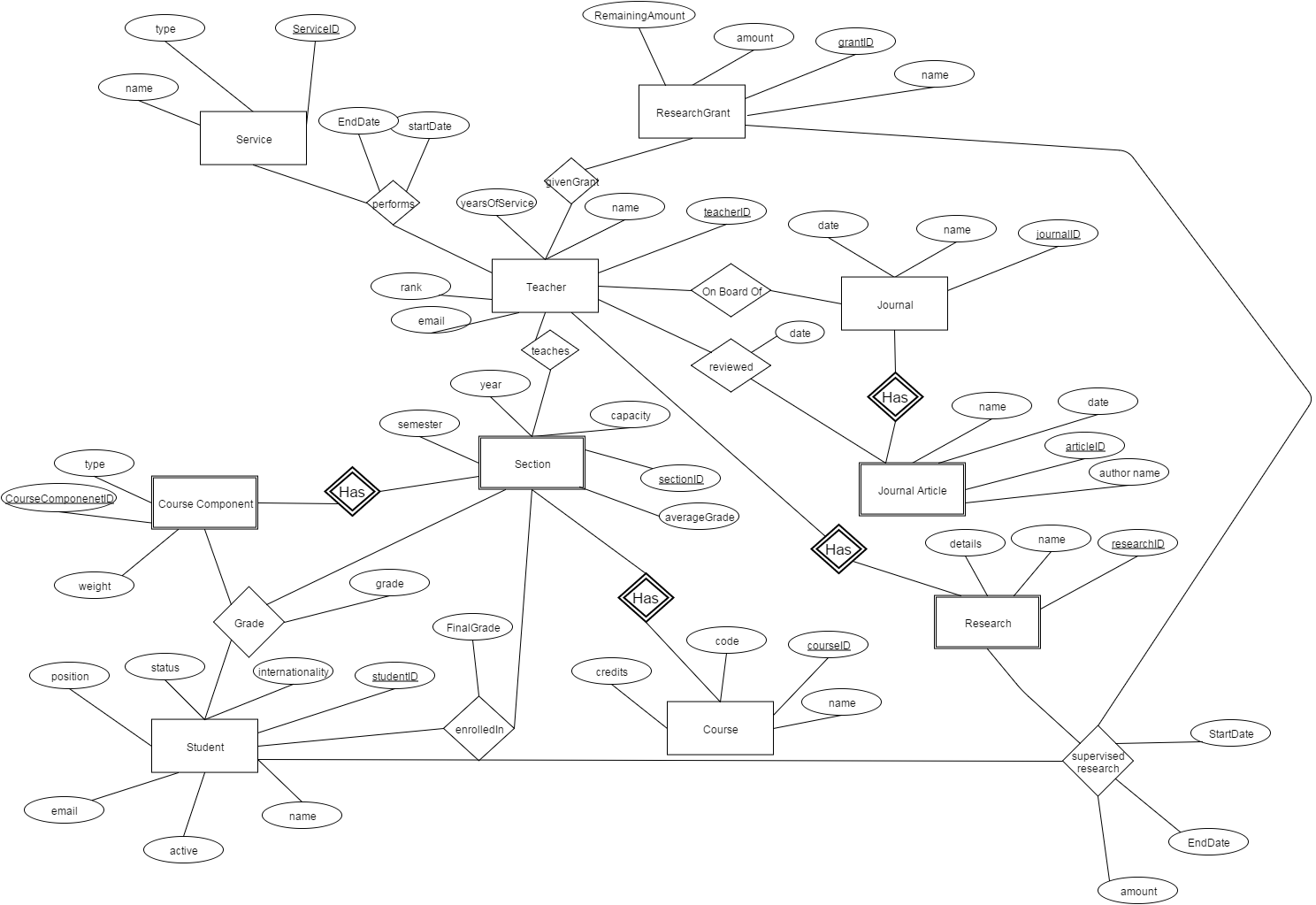
Each teacher can view their personal profile which displays their personal information as well as a list of committees, workshops, and conferences they attended along with their role in it.

The professor can also manage his journal contributions from the website. They can see the journals for which they are on the editorial board, as well as the articles they reviewed for that journal. All other articles the journal has published along with their relevant information are also available.

Teachers may view which research topics they are taking part in, the grants received for that research, and the grants received for all their research in total. The amount of grant money they have remaining, the students partaking in the research, and the funding the students received from the professor are displayed as well. The teacher may filter these students by year and/or position.

All the pages of the website scale properly on any mobile device (tablets, cellphones, etc..) and do not make use of any front-end or back-end framework.

**ER Diagram**



**ER to Relations Conversion**

Student(studentID, internationality, status, position, name, email, active)

Course(courseID, name, code, credits)

Section(sectionID, courseID, capacity, year, semester, averageGrade)

CourseComponent(courseComponentID, sectionID, CourseID, type, weight)

Grade(CourseComponentID, studentID, sectionID, courseID, grade)

EnrolledIn(studentID, sectionID, courseID, finalGrade)

Teacher(teacherID, password, email, name, yearsOfService, Rank)

Teaches(sectionID, teacherID, courseID)

Grant(grantID, amount, name, remainingAmount)

GivenGrant(teacherID, grantID)

Research(researchID, teacherID, name, details)

SupervisedResearch(studentID,researchID, teacherID, grantID, startDate, EndDate, amount)

Journal(journalID, name, date)

JournalArticle(articleID, journalID, authorName, name, date)

OnBoardOf(teacherID, journalID)

Reviewed(teacherID, articleID, journalID, date)

Service(serviceID, name, type)

performs(serviceID, teacherID, startDate, endDate)

**Normalization of Relations**

**Student(studentID, internationality, status, position, name, email, active)**

FD={studentID-> Student.internationality, Student.status, Student.position, Student.name, Student.active}

studentID+={studentID, internationality, status, position, name, email, active}

studentID a candidate key   
This relation is in BCNF, because the LHS of the nontrivial FD is a superkey.

**Course(courseID, name, code, credits)**

FDs={courseID-> Course.name, Course.code, Course.credits}

courseID+={courseID, name, code, credits}

FDs={Course.name ->courseID,Course.code, Course.credits}

Course.name+= {Course.name, courseID, code, credits}

FDs={Course.code-> courseID,name, credits}

Course.code+={Course.code, courseID,name, credits}

courseID a candidate key   
This relation is in 3NF, because the LHS of the nontrivial FD is a superkey (courseID).

RHS is a key attribute of Course (for course.name and course.code)

**Section(sectionID, courseID, capacity, year, semester, averageGrade)**

FDs={sectionID, courseID -> Section.capacity, Section.year, Section.Semester}

sectionID, courseID +={ sectionID, courseID,Section.capacity, Section.year, Section.Semester}

sectionID, courseID a candidate key   
This relation is in BCNF, because the LHS of the nontrivial FD is a superkey.

**CourseComponent(courseComponentID, sectionID, CourseID, type, weight)**

FDs={CourseComponenetID, sectionID, CourseID -> CourseComponent.type, CourseComponent.weight}

CourseComponenetID, sectionID, CourseID +={ CourseComponenetID, sectionID, CourseID, CourseComponent.type, CourseComponent.weight}

CourseComponenetID, sectionID, CourseID is a candidate key   
This relation is in BCNF, because the LHS of the nontrivial FD is a superkey.

**Grade(CourseComponentID, studentID, sectionID, courseID, grade)**

FDs={CourseComponentID, studentID, sectionID,courseID -> Grade.grade}

CourseComponentID, studentID, sectionID,courseID +={ CourseComponenetID, studentID, sectionID,courseID, Grade.grade}

CourseComponentID, sectionID, CourseID is a candidate key   
This relation is in BCNF, because the LHS of the nontrivial FD is a superkey.

**EnrolledIn(studentID, sectionID, courseID, finalGrade)**

FDs={studentID, sectionID , CourseID -> enrolledIn.FinalGrade}

studentID, sectionID,courseID +={ studentID, sectionID,courseID, enrolledIn.FinalGrade}

studentID, sectionID, CourseID is a candidate key   
This relation is in BCNF, because the LHS of the nontrivial FD is a superkey.

**Teacher(teacherID, password, email, name, yearsOfService, Rank)**

FDs={teacherID -> Teacher.name, Teacher.yearsOfService, Teacher.Rank}

teacherID +={ teacherID, Teacher.name, email, password, Teacher.yearsOfService, Teacher.Rank}

teacherID is a candidate key   
This relation is in BCNF, because the LHS of the nontrivial FD is a superkey.

**Teaches(sectionID, teacherID, courseID)**

FDs={sectionID, teacherID, courseID ->sectionID, teacherID, courseID}

sectionID, teacherID, courseID +={sectionID, teacherID, courseID}

sectionID, teacherID, courseID is a candidate key   
This relation is in BCNF, because the LHS of the nontrivial FD is a superkey.

**Grant(grantID, amount, name, remainingAmount)**

FDs = {grantID -> amount, name, remainingAmount}

grantID+= {grantID, amount, name, remainingAmount}.

This relation is in BCNF, because the LHS of the nontrivial FD is a superkey.

**GivenGrant(teacherID, grantID)**

FDs = teacherID, grantID -> {}

teacherID,grantID+= {teacherID,grantID}

This is in BCNF, because the LHS of the nontrivial FD is a superkey.

**Research(researchID, teacherID, name, details)**

FDs = {researchID, teacherID -> name, details}

researchID,teacherID+= {researchID,teacherID,name, details}

This is in BCNF, because the LHS of the nontrivial FD is a superkey.

**SupervisedResearch(studentID,researchID, teacherID, grantID, startDate, EndDate, amount)**

FDs={studentID, researchID, teacherID, grantID-> startDate, EndDate}

studentID, researchID, teacherID, grantID+ = {studentID, researchID, teacherID, grantID, startDate, EndDate}

This is in BCNF, because the LHS of the nontrivial FD is a superkey.

**Journal(journalID, name, date)**

FDs={journalID -> name, date}

journalID+={journalID,name,date}

This is in BCNF, because the LHS of the nontrivial FD is a superkey.

**JournalArticle(articleID, journalID, authorName, name, date)**

FDs = {articleID, journalID-> authorName, name, date}

articleID, journalID+= {authorName, name, date,articleID, journalID}

This is in BCNF, because the LHS of the nontrivial FD is a superkey.

**OnBoardOf(teacherID, journalID)**

FDs={teacherID, journalID -> {}}

teacherID, journalID + = {teacherID, journalID }

This is in BCNF, because the LHS of the nontrivial FD is a superkey.

**Reviewed(teacherID, articleID, journalID, date)**

FDs = {teacherID, articleID, journalID, ->date}

teacherID, articleID, journalID+ = {date,teacherID, articleID, journalID}

This is in BCNF, because the LHS of the nontrivial FD is a superkey.

**Service(serviceID, name, type)**

FDs ={serviceID -> name, type}

serviceID+ = {serviceID,name, type}

This is in BCNF, because the LHS of the nontrivial FD is a superkey.

**performs(serviceID, teacherID, startDate, endDate)**

FDs = {serviceID, teacherID -> startDate, endDate}

serviceID, teacherID += {serviceID, teacherID,startDate, endDate}

This is in BCNF, because the LHS of the nontrivial FD is a superkey.

**Create Tables**

Student(studentID, internationality, status, position, name, email, active, work)

CREATE TABLE IF NOT EXISTS Student(

studentID INT UNSIGNED NOT NULL AUTO\_INCREMENT PRIMARY KEY,

name VARCHAR(50) NOT NULL,

active ENUM('Yes','No') NOT NULL,

status ENUM('Full Time','Part Time','On academic probation','Failed standing','Suspended','Other') NOT NULL,

position ENUM('Undergraduate','Graduate','Masters','Phd','Other') NOT NULL,

internationality ENUM('Yes','No') NOT NULL,

email VARCHAR(50) NOT NULL,

work VARCHAR(50));

Course(courseID, name, code, credits)

CREATE TABLE IF NOT EXISTS Course(

courseID INT UNSIGNED NOT NULL AUTO\_INCREMENT PRIMARY KEY,

name VARCHAR(50) NOT NULL,

code VARCHAR(10) NOT NULL,

credits REAL UNSIGNED NOT NULL);

Section(sectionID, courseID, capacity, year, semester, averageGrade)

CREATE TABLE IF NOT EXISTS Section(

sectionID INT UNSIGNED NOT NULL AUTO\_INCREMENT,

courseID INT UNSIGNED NOT NULL,

semester ENUM('Summer','Fall','Winter') NOT NULL,

year INT(4) NOT NULL,

capacity INT(3) NOT NULL,

averageGrade INT NOT NULL,

FOREIGN KEY (courseID) REFERENCES Course(courseID)

ON DELETE CASCADE

ON UPDATE CASCADE,

PRIMARY KEY(sectionID, courseID));

CourseComponent(courseComponentID, sectionID, CourseID, type, weight)

CREATE TABLE IF NOT EXISTS CourseComponent(

courseComponentID INT UNSIGNED NOT NULL AUTO\_INCREMENT,

sectionID INT UNSIGNED NOT NULL,

courseID INT UNSIGNED NOT NULL,

type VARCHAR(50) NOT NULL,

weight INT UNSIGNED NOT NULL,

FOREIGN KEY (sectionID) REFERENCES Section(sectionID)

ON DELETE CASCADE

ON UPDATE CASCADE,

FOREIGN KEY (courseID) REFERENCES Section(courseID)

ON DELETE CASCADE

ON UPDATE CASCADE,

PRIMARY KEY(courseComponentID, sectionID, courseID));

Grade(CourseComponentID, studentID, sectionID, courseID, grade)

CREATE TABLE IF NOT EXISTS Grade(

grade INT NOT NULL,

studentID INT UNSIGNED NOT NULL,

sectionID INT UNSIGNED NOT NULL,

courseID INT UNSIGNED NOT NULL,

coursecomponentID INT UNSIGNED,

FOREIGN KEY (studentID) REFERENCES Student(studentID)

ON DELETE CASCADE

ON UPDATE CASCADE,

FOREIGN KEY (sectionID) REFERENCES CourseComponent(sectionID)

ON DELETE CASCADE

ON UPDATE CASCADE,

FOREIGN KEY (courseComponentID) REFERENCES CourseComponent (coursecomponentID)

ON DELETE CASCADE

ON UPDATE CASCADE,

FOREIGN KEY (courseID) REFERENCES CourseComponent (courseID)

ON DELETE CASCADE

ON UPDATE CASCADE,

PRIMARY KEY(CourseComponentID, StudentID, SectionID));

EnrolledIn(studentID, sectionID, courseID, finalGrade)

CREATE TABLE IF NOT EXISTS EnrolledIn(

finalGrade Enum('A+', 'A', 'A-', 'B+', 'B', 'B-', 'C+', 'C', 'C-', 'D+', 'D', 'D-', 'F', 'NA', 'DISC', 'DNE') NOT NULL,

studentID INT UNSIGNED NOT NULL,

sectionID INT UNSIGNED NOT NULL,

courseID INT UNSIGNED NOT NULL,

FOREIGN KEY (studentID) REFERENCES Student(studentID)

ON DELETE CASCADE

ON UPDATE CASCADE,

FOREIGN KEY (sectionID) REFERENCES Section(sectionID)

ON DELETE CASCADE

ON UPDATE CASCADE,

FOREIGN KEY (courseID) REFERENCES Section(courseID)

ON DELETE CASCADE

ON UPDATE CASCADE,

PRIMARY KEY(studentID, sectionID, courseID));

Teacher(teacherID, password, email, name, yearsOfService, Rank)

CREATE TABLE IF NOT EXISTS Teacher(

teacherID INT UNSIGNED NOT NULL AUTO\_INCREMENT PRIMARY KEY,

name VARCHAR(50) NOT NULL,

yearsOfService INT(3) UNSIGNED NOT NULL,

rank VARCHAR(100) NOT NULL,

password VARCHAR(50) UNIQUE NOT NULL,

email VARCHAR(50) NOT NULL );

Teaches(sectionID, teacherID, courseID)

CREATE TABLE IF NOT EXISTS Teaches(

teacherID INT UNSIGNED NOT NULL,

sectionID INT UNSIGNED NOT NULL,

courseID INT UNSIGNED NOT NULL,

FOREIGN KEY (teacherID) REFERENCES Teacher(teacherID)

ON DELETE CASCADE

ON UPDATE CASCADE,

FOREIGN KEY (sectionID) REFERENCES Section(sectionID)

ON DELETE CASCADE

ON UPDATE CASCADE,

FOREIGN KEY (courseID) REFERENCES Section(courseID)

ON DELETE CASCADE

ON UPDATE CASCADE,

PRIMARY KEY(teacherID, sectionID, courseID));

Grant(grantID, amount, name, remainingAmount)

CREATE TABLE IF NOT EXISTS ResearchGrant(

grantID INT UNSIGNED NOT NULL AUTO\_INCREMENT PRIMARY KEY,

name VARCHAR(100) NOT NULL,

amount INT UNSIGNED NOT NULL,

remainingAmount INT UNSIGNED NOT NULL);

GivenGrant(teacherID, grantID)

CREATE TABLE IF NOT EXISTS GivenGrant(

teacherID INT UNSIGNED NOT NULL,

grantID INT UNSIGNED NOT NULL,

FOREIGN KEY (teacherID) REFERENCES Teacher(teacherID)

ON DELETE CASCADE

ON UPDATE CASCADE,

FOREIGN KEY (grantID) REFERENCES ResearchGrant(grantID)

ON DELETE CASCADE

ON UPDATE CASCADE,

PRIMARY KEY(teacherID, grantID));

Research(researchID, teacherID, name, details)

CREATE TABLE IF NOT EXISTS Research(

researchID INT UNSIGNED NOT NULL AUTO\_INCREMENT,

name VARCHAR(50) NOT NULL,

details VARCHAR(25) NOT NULL,

teacherID INT UNSIGNED NOT NULL,

FOREIGN KEY (teacherID) REFERENCES Teacher(teacherID)

ON DELETE CASCADE

ON UPDATE CASCADE,

PRIMARY KEY (researchID, teacherID));

SupervisedResearch(studentID,researchID, teacherID, grantID, startDate, EndDate, amount)

CREATE TABLE IF NOT EXISTS SupervisedResearch(

teacherID INT UNSIGNED NOT NULL,

researchID INT UNSIGNED NOT NULL,

studentID INT UNSIGNED NOT NULL,

grantID INT UNSIGNED NOT NULL,

startDate DATE NOT NULL,

endDate DATE NOT NULL,

amount INT UNSIGNED NOT NULL,

FOREIGN KEY (teacherID) REFERENCES Research(teacherID)

ON DELETE CASCADE

ON UPDATE CASCADE,

FOREIGN KEY (studentID) REFERENCES Student(studentID)

ON DELETE CASCADE

ON UPDATE CASCADE,

FOREIGN KEY (researchID) REFERENCES Research(researchID)

ON DELETE CASCADE

ON UPDATE CASCADE,

FOREIGN KEY (grantID) REFERENCES ResearchGrant(grantID)

ON DELETE CASCADE

ON UPDATE CASCADE,

PRIMARY KEY(teacherID, studentID, researchID, grantID));

Journal(journalID, name, date)

CREATE TABLE IF NOT EXISTS Journal(

journalID INT UNSIGNED NOT NULL AUTO\_INCREMENT PRIMARY KEY,

name VARCHAR(100) NOT NULL,

date DATE NOT NULL);

JournalArticle(articleID, journalID, authorName, name, date)

CREATE TABLE IF NOT EXISTS JournalArticle(

articleID INT UNSIGNED NOT NULL AUTO\_INCREMENT,

journalID INT UNSIGNED NOT NULL,

name VARCHAR(100) NOT NULL,

authorName VARCHAR(50) NOT NULL,

date DATE NOT NULL,

FOREIGN KEY (journalID) REFERENCES Journal(journalID)

ON DELETE CASCADE

ON UPDATE CASCADE,

PRIMARY KEY(articleID, journalID));

OnBoardOf(teacherID, journalID)

CREATE TABLE IF NOT EXISTS OnBoardOf(

journalID INT UNSIGNED NOT NULL,

teacherID INT UNSIGNED NOT NULL,

FOREIGN KEY (teacherID) REFERENCES Teacher(teacherID)

ON DELETE CASCADE

ON UPDATE CASCADE,

FOREIGN KEY (journalID) REFERENCES Journal(journalID)

ON DELETE CASCADE

ON UPDATE CASCADE,

PRIMARY KEY (journalID, teacherID));

Reviewed(teacherID, articleID, journalID, date)

CREATE TABLE IF NOT EXISTS Reviewed(

teacherID INT UNSIGNED NOT NULL,

articleID INT UNSIGNED NOT NULL,

journalID INT UNSIGNED NOT NULL,

date DATE NOT NULL,

FOREIGN KEY (teacherID) REFERENCES Teacher(teacherID)

ON DELETE CASCADE

ON UPDATE CASCADE,

FOREIGN KEY (articleID) REFERENCES JournalArticle(articleID)

ON DELETE CASCADE

ON UPDATE CASCADE,

FOREIGN KEY (journalID) REFERENCES JournalArticle(journalID)

ON DELETE CASCADE

ON UPDATE CASCADE,

PRIMARY KEY(teacherID, articleID, journalID));

Service(serviceID, name, type)

CREATE TABLE IF NOT EXISTS Service(

serviceID INT UNSIGNED NOT NULL AUTO\_INCREMENT PRIMARY KEY,

name VARCHAR(50) NOT NULL,

type VARCHAR(100) NOT NULL);

Performs(serviceID, teacherID, startDate, endDate)

CREATE TABLE IF NOT EXISTS Performs(

serviceID INT UNSIGNED NOT NULL,

teacherID INT UNSIGNED NOT NULL,

startDate DATE NOT NULL,

endDate DATE NOT NULL,

FOREIGN KEY (teacherID) REFERENCES Teacher(teacherID)

ON DELETE CASCADE

ON UPDATE CASCADE,

FOREIGN KEY (serviceID) REFERENCES Service(serviceID)

ON DELETE CASCADE

ON UPDATE CASCADE,

PRIMARY KEY (serviceID, teacherID));

**Triggers**

//after inserting supervisedresearch update remaining amount of grant

delimiter $$

CREATE TRIGGER UpdateAmount After INSERT ON SupervisedResearch

FOR EACH ROW

BEGIN

Update ResearchGrant

SET remainingAmount = (remainingAmount - NEW.amount)

WHERE grantID = NEW.grantID;

END;

$$

//before inserting supervised research check if grant has sufficient amount

//if student is an undergrad check that his research is in summer only

delimiter $$

CREATE TRIGGER remainingGrantAmount BEFORE INSERT ON SupervisedResearch

FOR EACH ROW

BEGIN

IF (NEW.amount > (SELECT remainingAmount FROM ResearchGrant WHERE grantID = NEW.grantID)) THEN

SET @MESSAGE = 'NO SUFFICIENT FUNDS';

SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT=@MESSAGE;

END IF;

IF ('UNDERGRADUATE' = (SELECT position FROM Student WHERE studentID = NEW.studentID)) THEN

IF(NOT(month(NEW.startDate) >= 6 and month(NEW.endDate) <= 9 and year(NEW.startDate) = year(NEW.endDate)) ) THEN

SET @MESSAGE = 'undergraduate student can not do research except in summer';

SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT=@MESSAGE;

END IF;

END IF;

END;

$$

**Queries**

\*ID when given is represented by value of 1

1. SELECT semester FROM Section

WHERE (sectionID, courseID) IN

{ SELECT s.sectionID, s.courseID FROM Section s, Teaches t, Course c

WHERE t.teacherID=1 AND c.courseID=t.courseID

AND t.sectionID =s.sectionID AND s.courseID=c.courseID

AND c.name=”Databases”}

2. SELECT c.name FROM Course c, Teaches t, Section s

WHERE t.teacherID=1 AND s.courseID=c.courseID AND s.courseID=t.courseID AND s.sectionID=t.sectionID AND ((s.year=X.year AND s.semester>=X.n) OR (s.year=Y.year AND s.semester<=Y.n) OR (s.year>X.year AND s.year <Y.year));

\*Assume X and Y are objects containing year and semester

3. SELECT s.name FROM Student s, SupervisedResearch r

WHERE r.teacherID=1 and s.studentID=r.studentID AND s.position=”Masters”;

4.

(SELECT COUNT(distinct studentID)

FROM SupervisedResearch

WHERE (startdate <= '"$Student\_finished\_research."' AND

enddate >='".$Student\_start\_research') AND

StudentCategory = $StudentCategoryWanted ")

5. SELECT SUM(r.amount) AS “Total Support Given”

FROM Student s, SupervisedResearch r

WHERE s.position=”Graduate” AND s.studentID=1 AND r.studentID=s.studentID

GROUP BY s.studentID;

6. SELECT COUNT(s.studentID) AS “Total Students Taught”

FROM Student s, Teaches t

WHERE t.teacherID=1 AND s.studentID=t.studentID

GROUP BY s.studentID;

7. SELECT Section.sectionID , Section.courseID, Section.semester, Section.year

FROM teaches, Section

WHERE teaches.teacherID = 4 AND teaches.sectionID = Section.sectionID

AND teaches.courseID = Section.courseID

AND Section.averageGrade = (SELECT MAX(averagegrade) FROM Section s , Teaches t WHEREt.teacherID = 4 and s.courseID = t.courseID and s.sectionID = t.sectionID)

8.SELECT \* FROM Performs, Service WHERE Performs.teacherID = $TeacherID\_Wanted AND Service.type =’committee member’ AND perform.serviceID = Service.serviceID AND perform.startDate >$date AND perform.EndData < $date2

**Extra**

* //select all the sections taught by a teacher with the number of students enrolled in + filtering by date and course code or any of them.

"SELECT Course.courseID as courseID, Course.name as courseName,

Course.code as courseCode, Course.credits as courseCredits,

Section.sectionID as sectionID, Section.semester as sectionSemester,

Section.year as sectionYear, Section.capacity as sectionCapacity,

Section.averageGrade as AverageGrade,

(select count(\*)

FROM enrolledin

WHERE enrolledin.courseID = Course.courseID

and enrolledin.sectionID = Section.sectionID)

as enrolled

FROM Course, Section

WHERE Section.courseID = Course.courseID";

if(isset($\_GET['fy']) ? (int) $\_GET['fy'] : null)

{

{$sql = $sql. " AND section.year >= ".$\_GET['fy'];}

}

if(isset($\_GET['ty']) ? (int) $\_GET['ty'] : null)

{$sql = $sql. " AND section.year <= ".$\_GET['ty'];}

if(isset($\_GET["cn"]))

{$sql = $sql. " AND Course.code = '".$\_GET['cn']."'";}

$sql = $sql ." AND (Course.courseID , Section.sectionID)

IN (SELECT courseID, sectionID

FROM Teaches

WHERE teacherID = ".$\_SESSION['teacherID'].");

";

* //select section info + number of students enrolled of given sectionID and courseID

"SELECT Course.courseID as courseID,

Course.name as courseName, Course.code as courseCode,

Course.credits as courseCredits,

Section.sectionID as sectionID, Section.semester as sectionSemester,

Section.year as sectionYear, Section.capacity as sectionCapacity,

(select count(\*)

FROM enrolledin

WHERE enrolledin.courseID = ".$courseID." AND enrolledin.sectionID =" .$sectionID.")

AS enrolled

FROM Course, Section

WHERE Course.courseID = " .$courseID. " AND Section.sectionID = " .$sectionID." AND Section.courseID = Course.courseID;";

* //select students enrolled in a section + their grade

SELECT

E.studentID AS sID, E.FinalGrade as sFinalGrade, S.name AS sName,

S.status AS sStatus, S.position AS sPosition,

S.internationality AS sInternationality, S.email as sEmail

FROM Enrolledin AS E, Student AS S

WHERE E.CourseID =".$courseID."

AND E.SectionID = ".$sectionID." AND E.studentID = S.StudentID;

//Gets all courses taught by prof

* SELECT c.code,c.name FROM Teaches t, Course c WHERE t.courseID=c.courseID

AND t.teacherID=1;

//Gets student info

* SELECT \* FROM Student WHERE studentID=1

//Gets course info that student has taken

* SELECT c.code,c.name,e.finalGrade,e.sectionID,e.courseID FROM EnrolledIn e, Course c WHERE e.courseID=c.courseID AND e.studentID=1

//Gets teachers research information

* SELECT Research.researchID,Research.name,Research.details FROM Research,Teacher WHERE Teacher.teacherID=$tid and Teacher.teacherID=Research.teacherID

//Gets teachers grants information

* SELECT grantID, name, amount, RemainingAmount

FROM ResearchGrant

Where grantID IN (SELECT grantID from GivenGrant where teacherID=".$tid.")

//Gets students participating in a research

* SELECT (Select name from student where SupervisedResearch.studentID=student.studentID) as Name, startDate, endDate, amount

FROM SupervisedResearch

WHERE SupervisedResearch.researchID=$nb)

//Gets Grant Information

* Select grantID, amount, name, RemainingAmount from ResearchGrant where

(SELECT grantID FROM SupervisedResearch where SupervisedResearch.researchID=$nb ORDER BY grantID LIMIT 1)=ResearchGrant.grantID")

//Gets all students doing research for professor and filter by Year and Position

* SELECT name,position,studentID

FROM Student

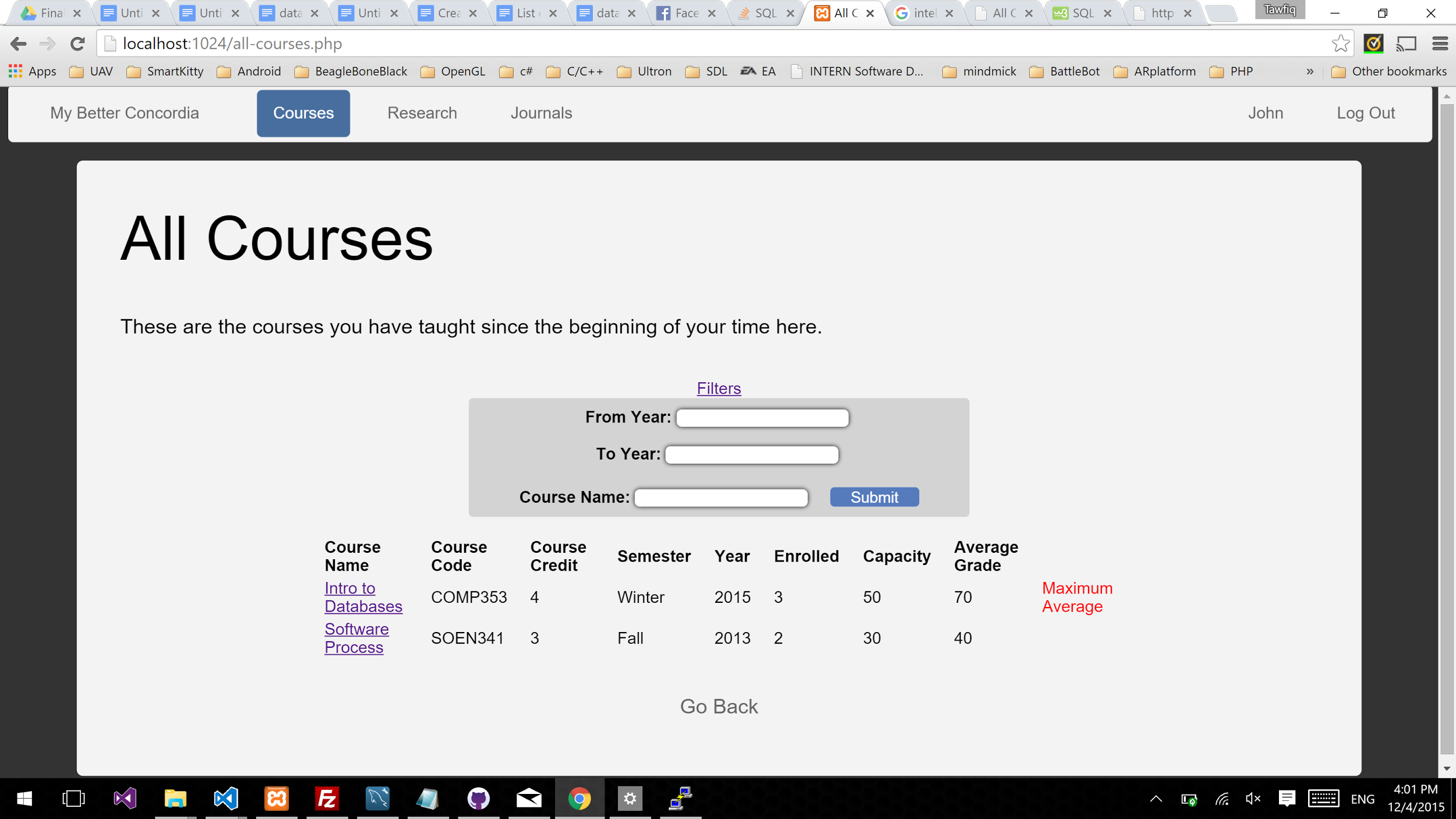
WHERE studentID IN(SELECT distinct studentID FROM SupervisedResearch WHERE teacherID=$tid $datefilter ORDER BY studentID) $positionfilter

$datefilter= " AND (startdate <= '".$inc\_to."' AND enddate >='".$inc\_from."') ";

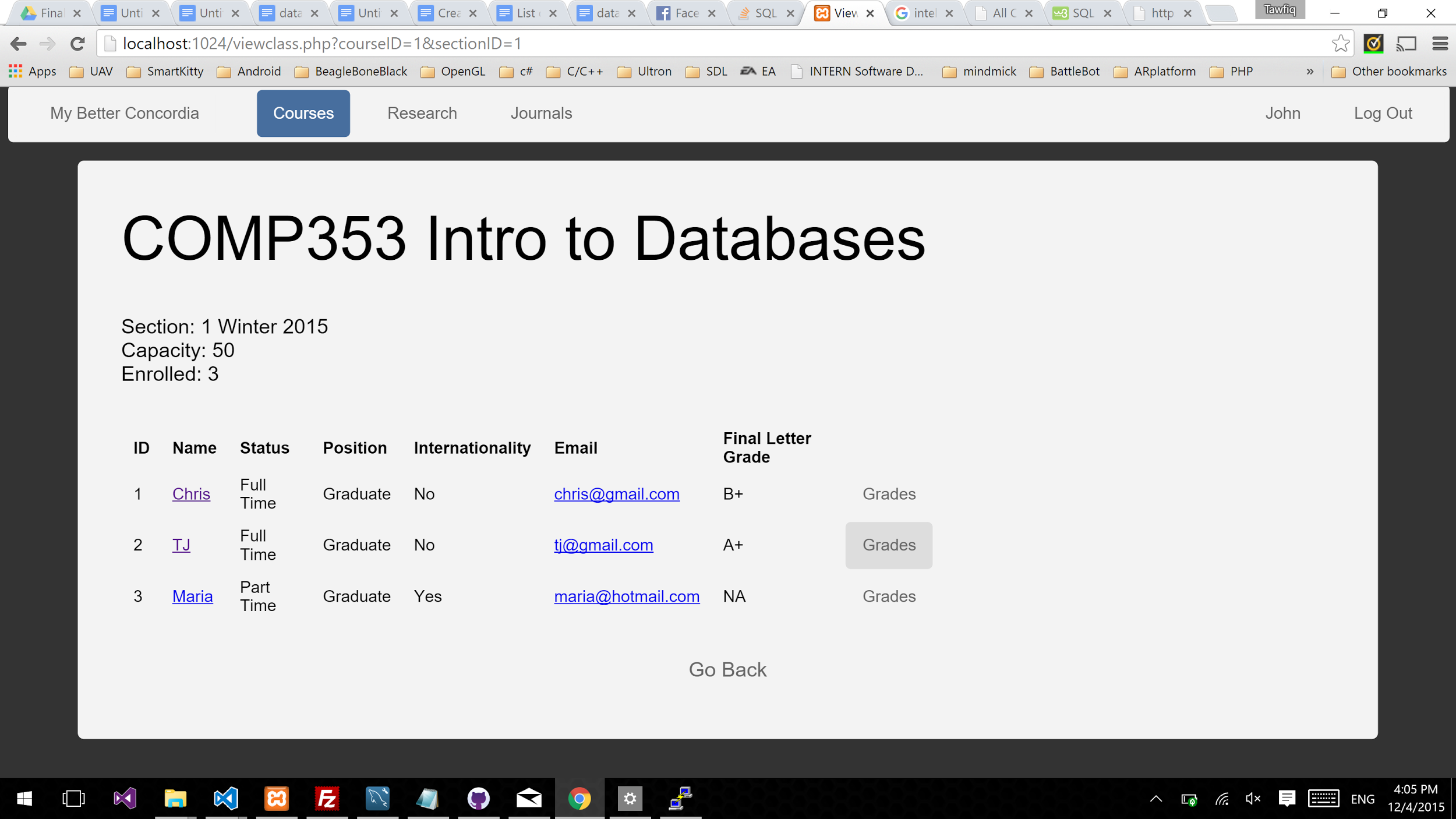
$positionfilter=" AND position = '".$pos."'";

**Sample Output**

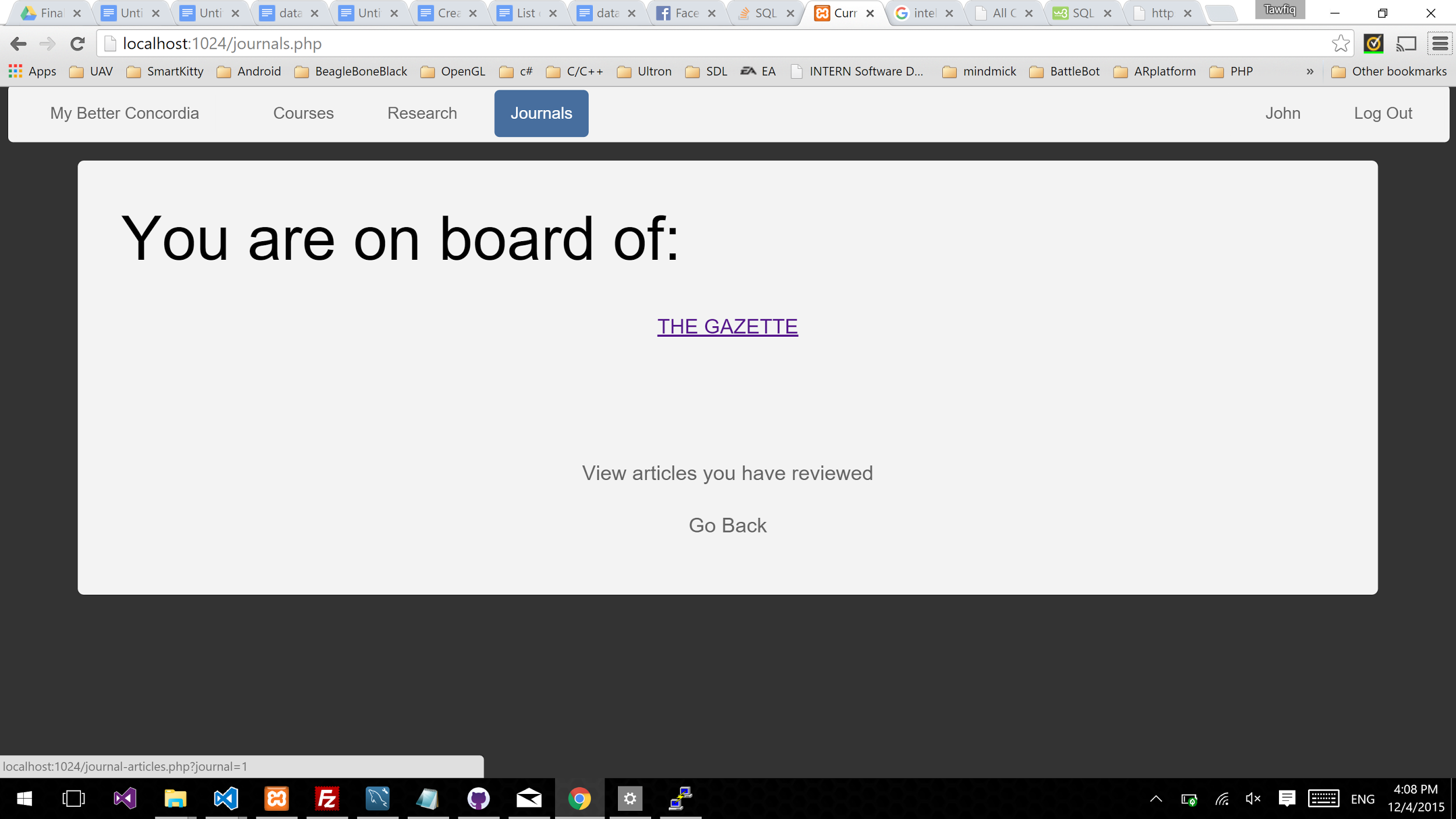
Course Page:



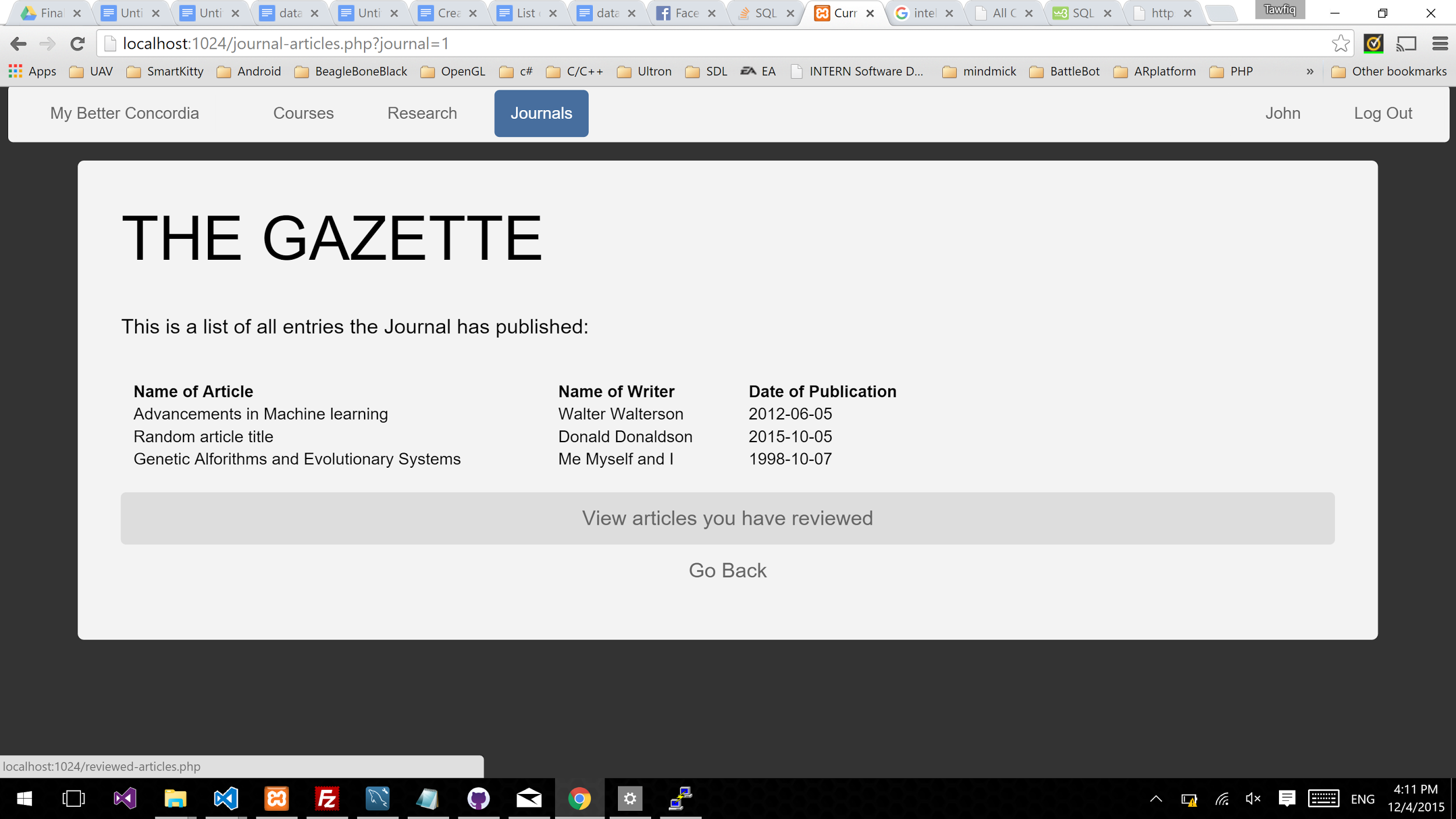
Section Page: displays the students enrolled in the section and their grades for every course component. (for example Midterm, Assignment, Final Exam)

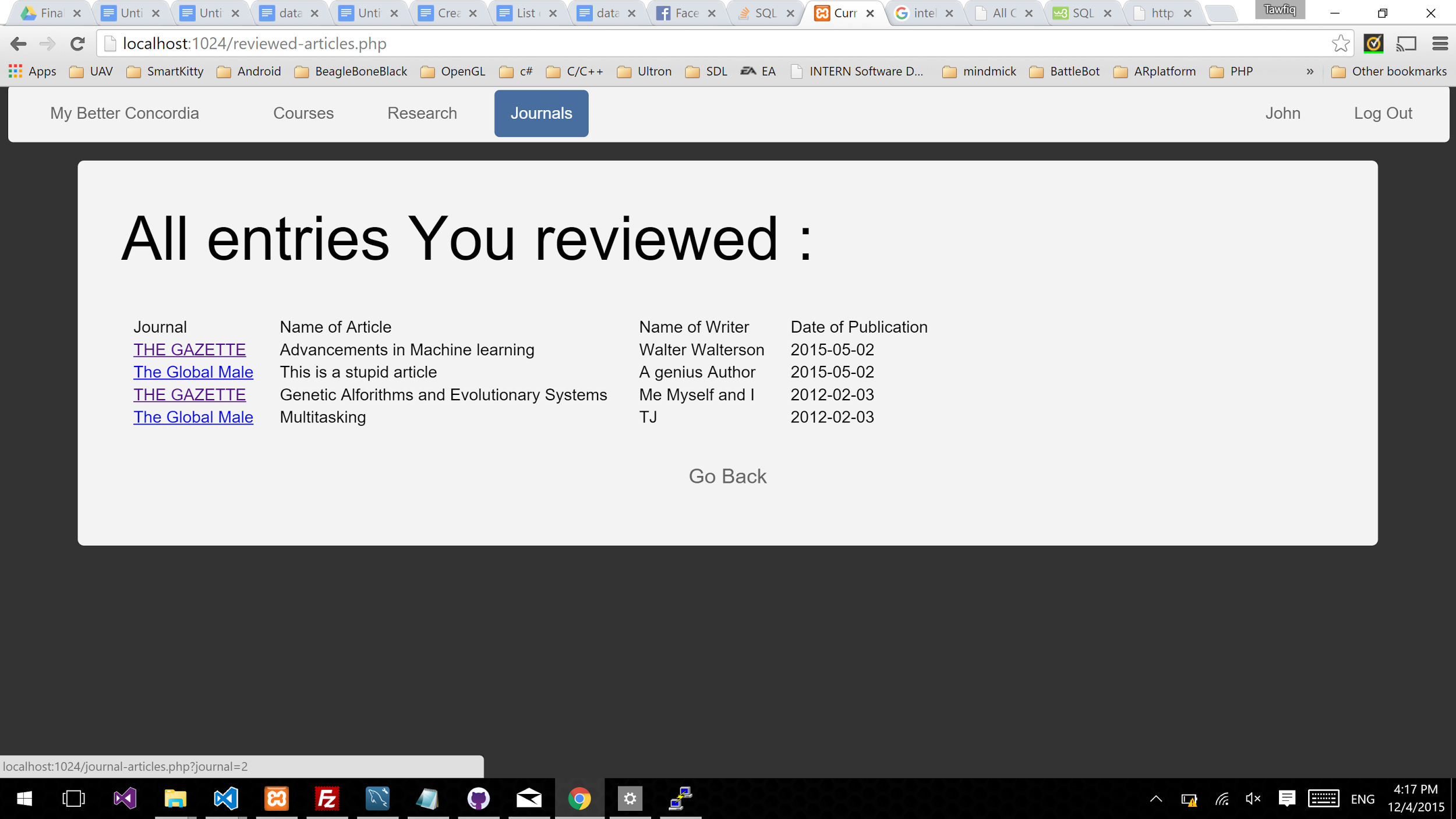


Journal On Board Of Page: displays the journals the user is on board of.

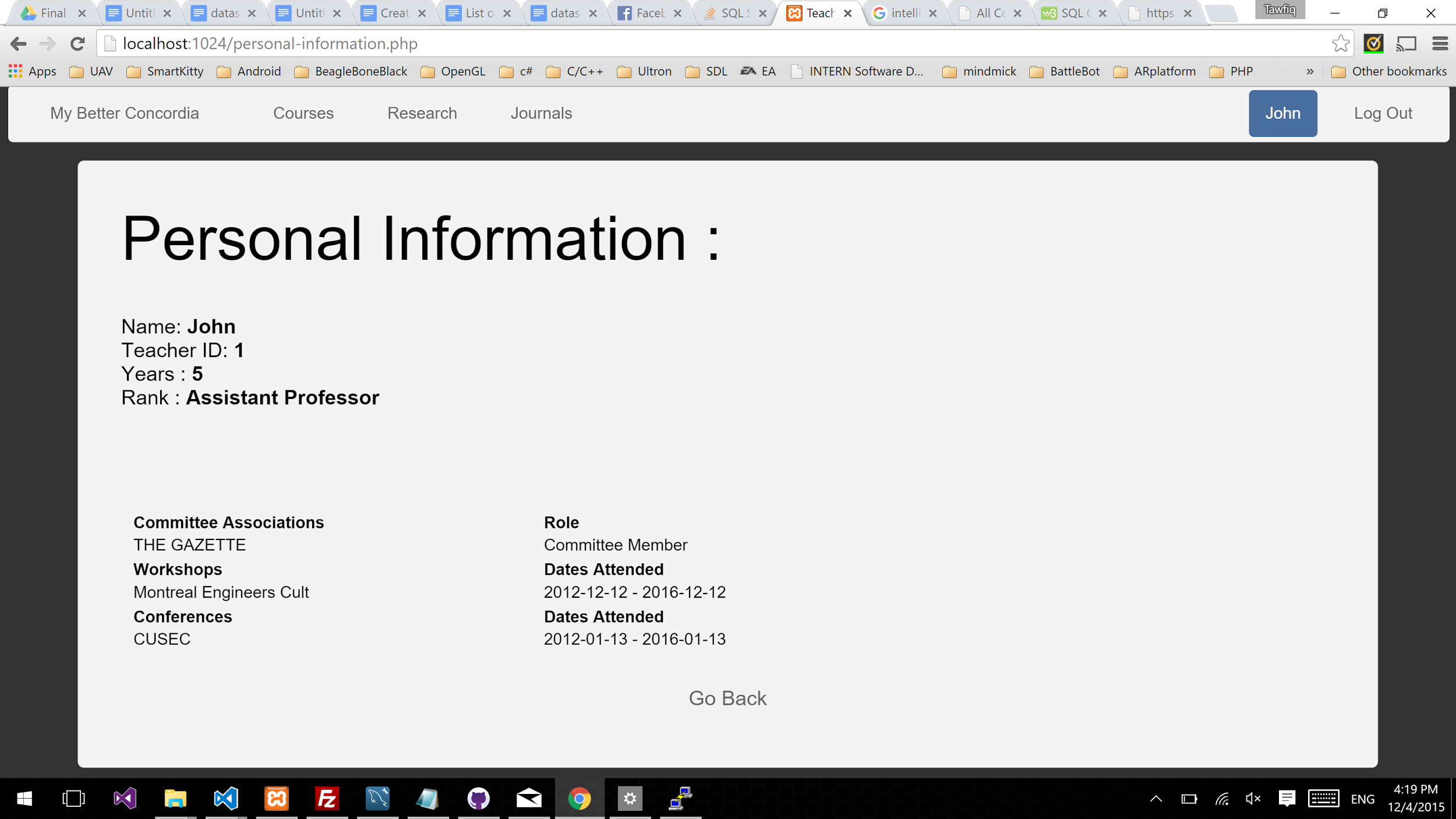


Journal Page: Displays all articles of a Journal

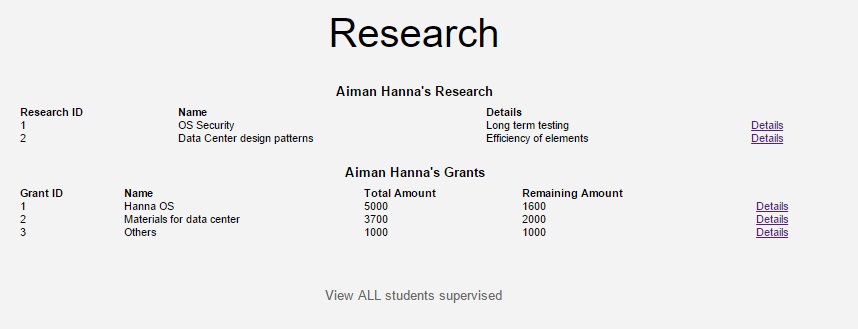


Journal Page: View articles you have reviewed.

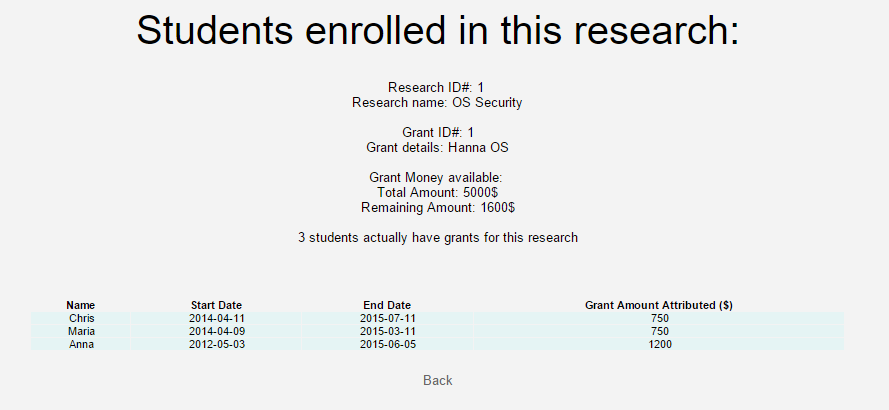
Personal Information: Display the user’s information and services



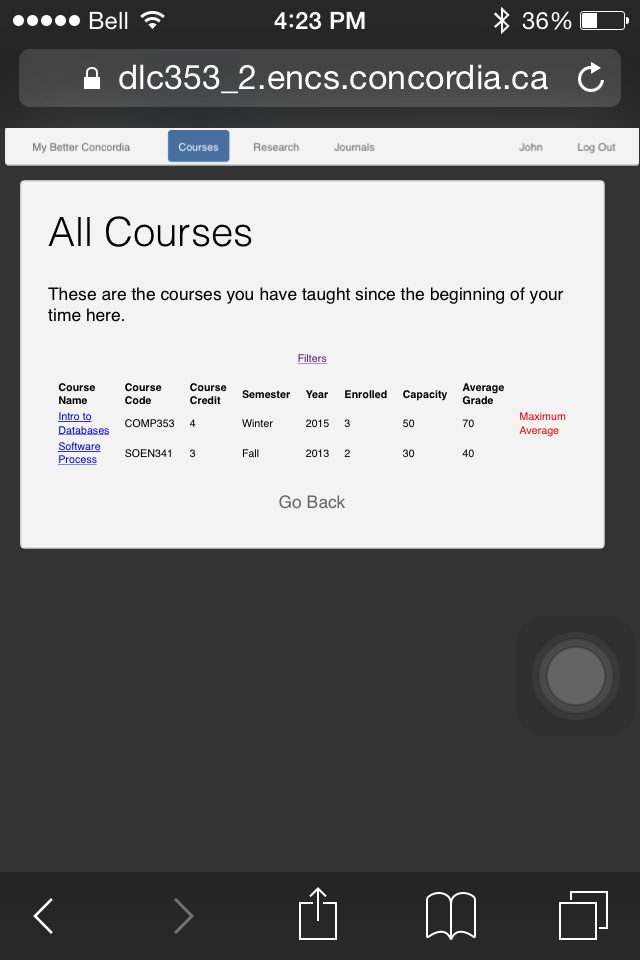
Research Page: Displays lists of the logged in teacher research and grant



Lists the students participating in a particular research and details about its grant.



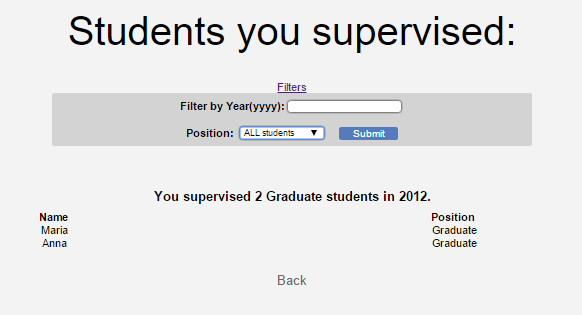
Mobile



Displays the researches linked to a particular grant



Displays all students supervised teacher and filters on the year and Position(Graduate, phd…)



**Contributions**

Christiano : Managed and created the login system and system navigation. Created the teacher and student information pages. Drafted original ER diagram and converted it into relations. Converted relations to BCNF using derived FDs. Dealt with both front-end html,css, javascript, jquery and back-end php.

Tawfiq: Managed and created the course and journal pages with filter queries. Managed the concordia server. Drafted original ER diagram and converted into relations and worked on FDs. Converted relations to BCNF using derived FDs.

Andrei: Worked on the Research and Grant pages. Built the first version of the sql tables. And draw the first ER diagram from our group discussions and handwritten draft.

Anuvert: Created the front-end of the website. Designed the UI for all of the pages. Created the list of queries. Helped set up the group folder given to us by Stan.

