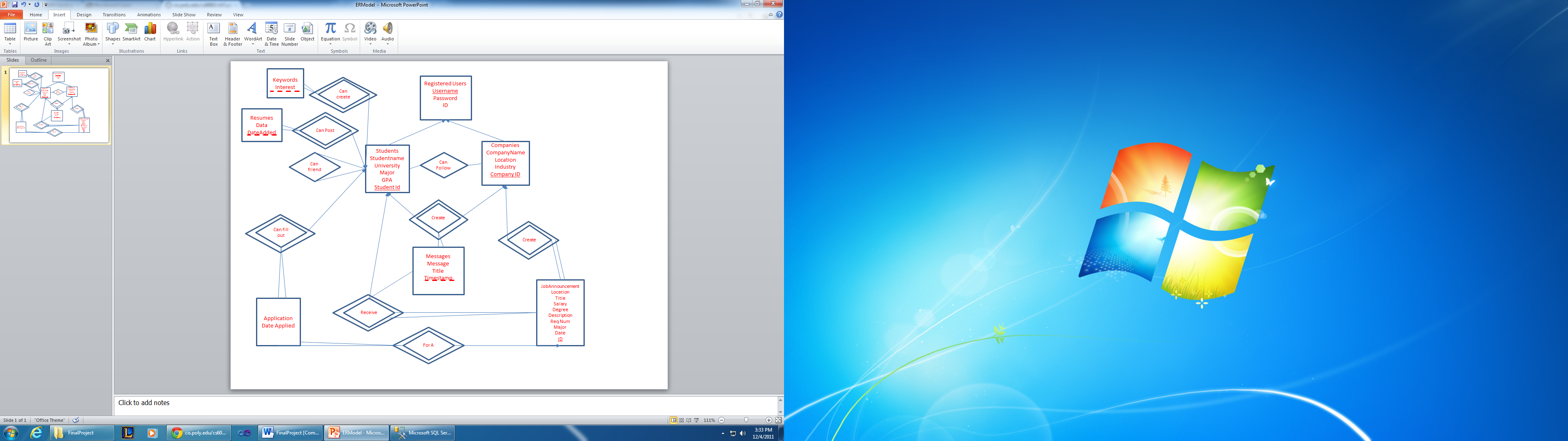
**Introduction**

The goal of this project is to design and create a fully functional job search site called Jobwalla. Jobwalla will be driven by a backend database schema with a php frontend hosted on a web server. This site will allow companies and students to register and provide a way for them to interact and contact each other for job opportunities. Students can follow companies and friend other students to share company information. Jobwalla will provide the following features.

1. Allow students to sign up
2. Allow students to create a personal profile
3. Allow students to post resumes
4. Allow companies to sign up
5. Allow companies to post a personal profile.
6. Students can follow companies (receive all their announcements)
7. Students can apply for jobs
8. Students can friend students
9. Students can message students
10. Companies can create announcements
11. Students can forward announcements
12. Companies can send announcements to students
13. Only friends can see students full profiles
14. Only companies which the student applied can see their resume
15. Students can see company profiles
16. Allow companies to search for students on certain criteria
17. Allow students to search for announcements based on criteria

The database backend, created in SQL Server 2008, will serve as the relational model to the PHP front end site. We start this process by designing an ER model based on the requirements provided above and translate this model into a relational schema.

**ER Model (Part A)**



This ER model shows the students and companies table being the main focus of the project. All student features will require a link to the students table. Similarly all company features require a link to the company table. We see that both students and companies are of type registered users because both are allowed to register on the site.

At a high level we can see clearly, that this ER model supports all the features listed in the introduction.

**Relational Schema (Part A)**

Below are the tables with keys, foreign keys and data types based on the ER model. We link these tables to the features listed in the introduction, to ensure that we fully meet all the requirements.

Note: varchar(max) = CLOB in sql server.

**Goal B and E. (Goals O and P achieved by the front end through these tables)**

First, we simply create the student and company tables. These are two straightforward tables.

**Students(StudentName(varchar), University(varchar), Major(varchar), GPA(float), StudentId(int))**

**Companies(CompanyId(int), CompanyName(varchar), Location(varchar), Industry(varchar))**

**Goal A and D**

Next we create two separate tables for the registered students and registered company users. I decided to separate these into two tables, because it will allow for separate registration pages based on the type of person registering. Although the registration information is the same, the id that maps the login to the profile is different and we want to distinguish between the two. We can also simply create a stored procedure later which automatically inserts and links the two records together when a user registers on the site.

**RegisteredStudents(Username(varchar), Password(varchar), StudentId(int))**

**StudentId references Students**

**RegisteredCompanies(Username(varchar), Password(varchar), CompanyId(int))**

**CompanyId references Companies**

**Goal C**

Next we create the table for resumes. Note that a varchar(max) is the same as a CLOB. We also create a new table called keywords. Although we want to search the resume clob using like and contains, we also create the keyword table to allow students to enter their own preferences as well.

**Resumes(StudentId(int), Resume(nvarchar(max)), DateAdded(Date), resumeId(int))**

**StudentId references Student**

**StudentKeywords(StudentId(int), Interest(varchar))**

**StudentId references Students**

**Goal F**

Students can follow companies so we model this simply using the ids in a table.

**StudentFollows(StudentId(int), CompanyId(int))**

**StudentId references Students, CompanyId references Companies**

**Goal H and M**

Friendship is modeled the same except we can confirm or deny a friendship. It is a two way agreement. Therefore we have a table for the requests and a table for the friendships. When a friendship is agreed too, a stored procedure can add the two tuples into the friendship table to model the two way friendship and delete the request from the table. We can model the limited profile by displaying different data to friends and non friends through either a query, or a separate view.

**Friendship(StudentId1(int), StudentId2(int))**

**StudentId1 and StudentId2 references Students**

**FriendRequests(StudentRequestedId(int), StudentIntendedId(int), DateRequested(Date))**

**StudentRequestedId and StudentIntendedId References Students**

**Goal J and K and L (Goal Q achieved through the front end on this table)**

We now model the job announcements. We have a simple table which lists all the announcements that a company can make.

**JobAnnouncements(CompanyId(int), JobLocation(varchar), Jobtitle(varchar), Salary(money), Degree(varchar), Description(varchar(max)), RequisitionNumber(varchar), AnnouncementId(int), Major(varchar), DatePosted(Date))**

**CompanyId references Companies**

However, this is not enough information to model the fact that some students may wish to save announcements or receive announcements from other students. We wish to create an inbox just for company announcements. We can do this using this table.

**AnnouncementsForStudents(StudentId(int), AnnouncementId(int))**

**StudentId references Students**

**Goal I**

We wish to model messaging. Students can message students. And companies can message students. This will be a separate inbox in the database. If time permits, we can model whether a message is read or unread, for the time being we will leave that part alone and simply model the message.

**StudentMessages(OrigStudentId(int), RecptStudentId(int), Message(varchar), Timestamp(DateTime), Title(varchar))**

**OrigStudentId and RecptStudentId references Students**

**CompanyMessages(CompanyId(int), StudentId(int), Message(varchar), Timestamp(datetime), Title(varchar))**

**CompanyId references Companies, StudentId references Students**

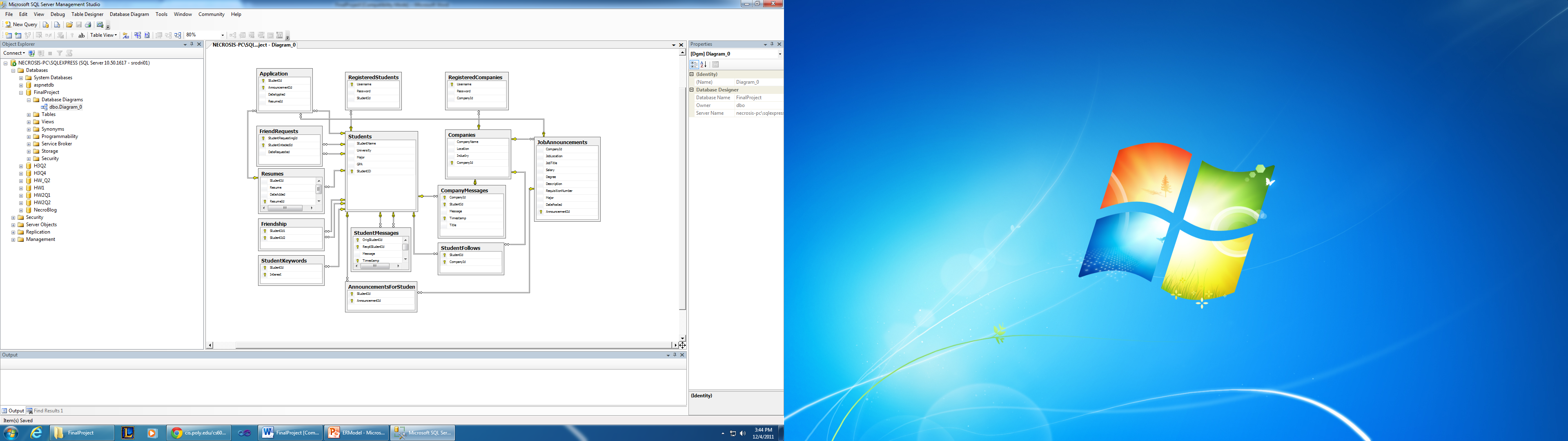
**Goal G and N**

We wish to model students applying for jobs. We do this by linking the student id, with the announcement id and a specific resume they wish for the company to see. This way, the company now has access through this table to a student’s resume.

**Application(StudentId(int), AnnouncementId(int), DateApplied(Datetime), ResumeId(Int))**

**Creating the Database in SQL Server**

I have used SQL server 2008 to create the database and its tables. Below is a SQL Server diagram of the database layout.



All of the above tables were created as per the schema shown in the previous section. We can see that the registered students/companies tables are now two separate entities to allow for both to register separately.**Testing – Sample Data and Queries**

Below are the initial tests required by Project #1 assignment and the results.

1. Create a record for a new student account, with a name, a login name, and a password

BEGIN TRAN

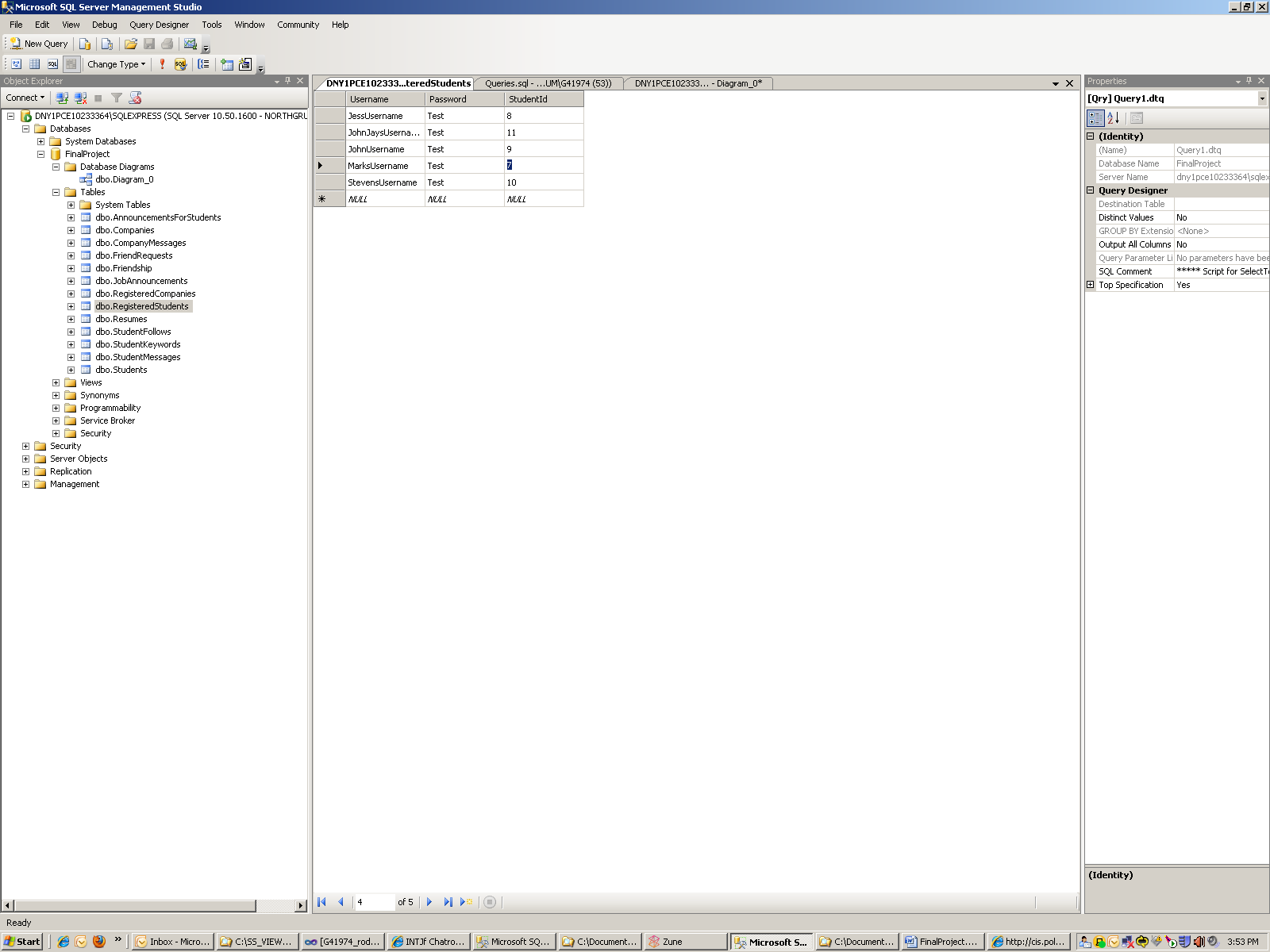
INSERT INTO Students VALUES ('Mark Jones', 'NYU', 'CS', '3.6');

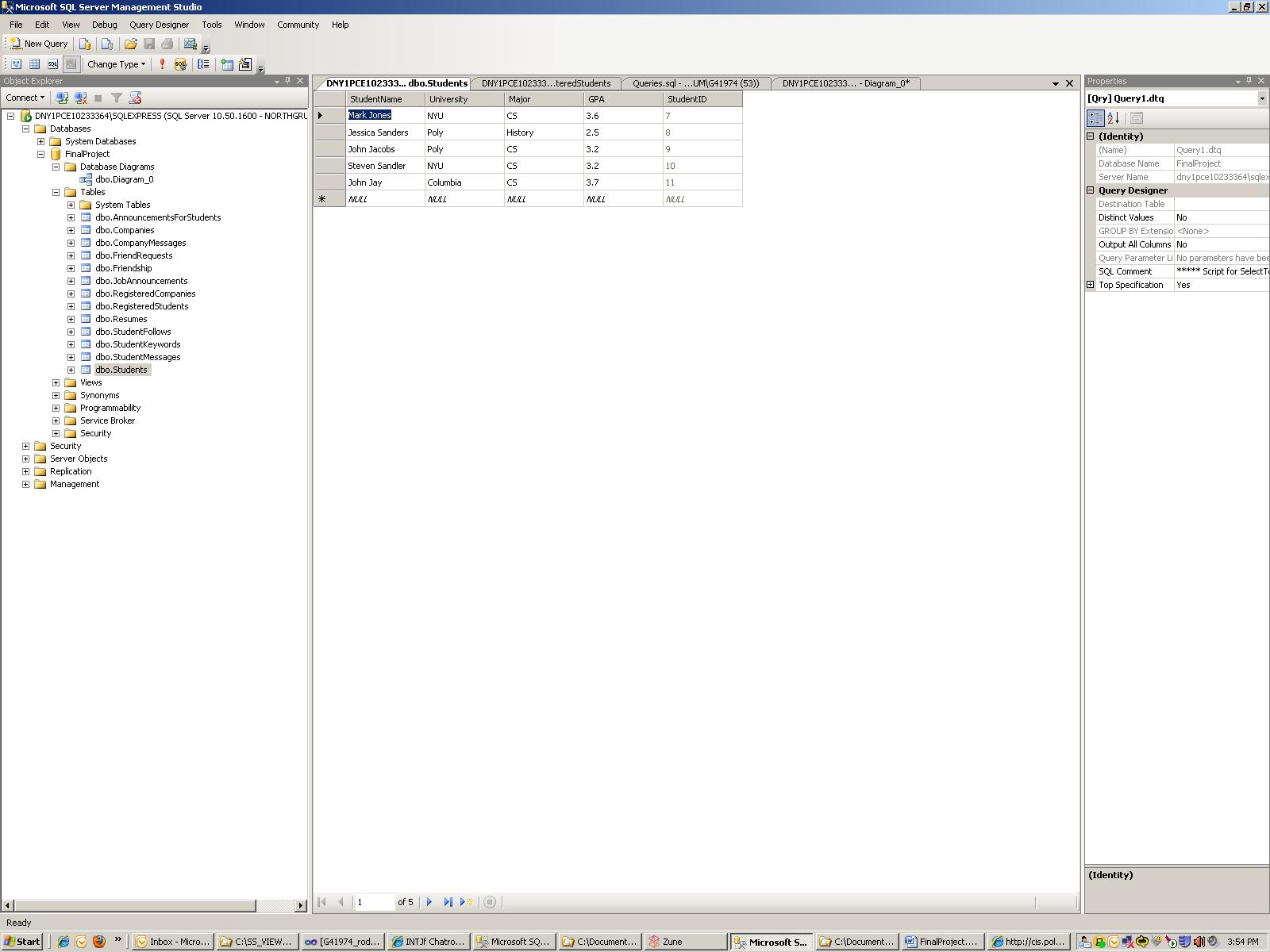
DECLARE @val int;

SET @val = (select max(StudentId) from Students)

INSERT INTO RegisteredStudents VALUES ('MarksUsername', 'Test', @val);

COMMIT TRAN





As you can see, if we execute the query in part A, we successfully create the Student Mark Jones and the matching registered student with his username and password. We do this for 4 more students. Two students with GPA’s over 3.5 to help give valid results for query E. We also create several students at NYU to help with query D.

(B)

List the names of all friends of a particular user

Select Students.StudentName from dbo.Students

where StudentId in (select StudentId2 from dbo.Friendship

where StudentId1 in

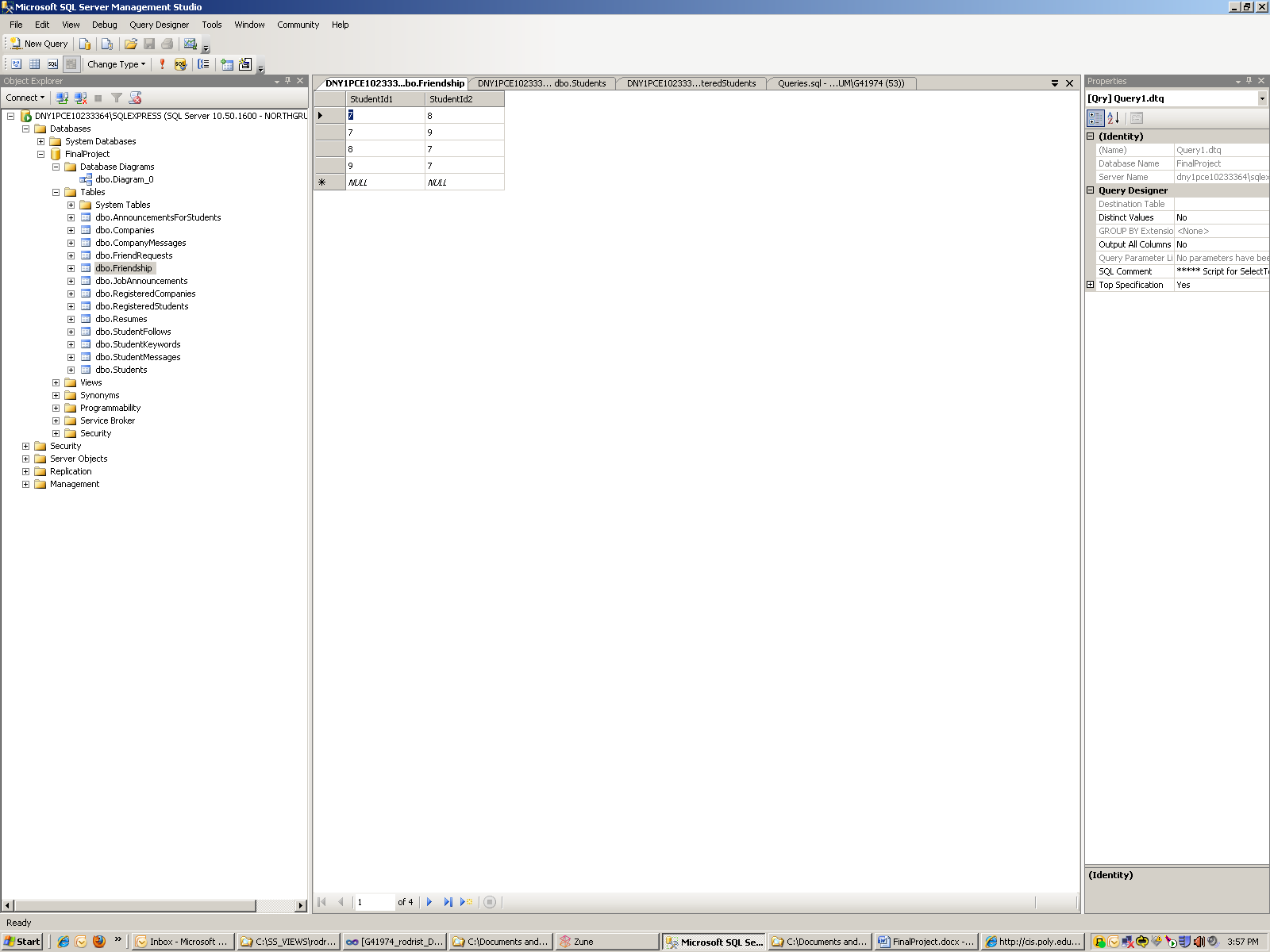
(select StudentId from dbo.Students

where StudentName = 'Mark Jones')

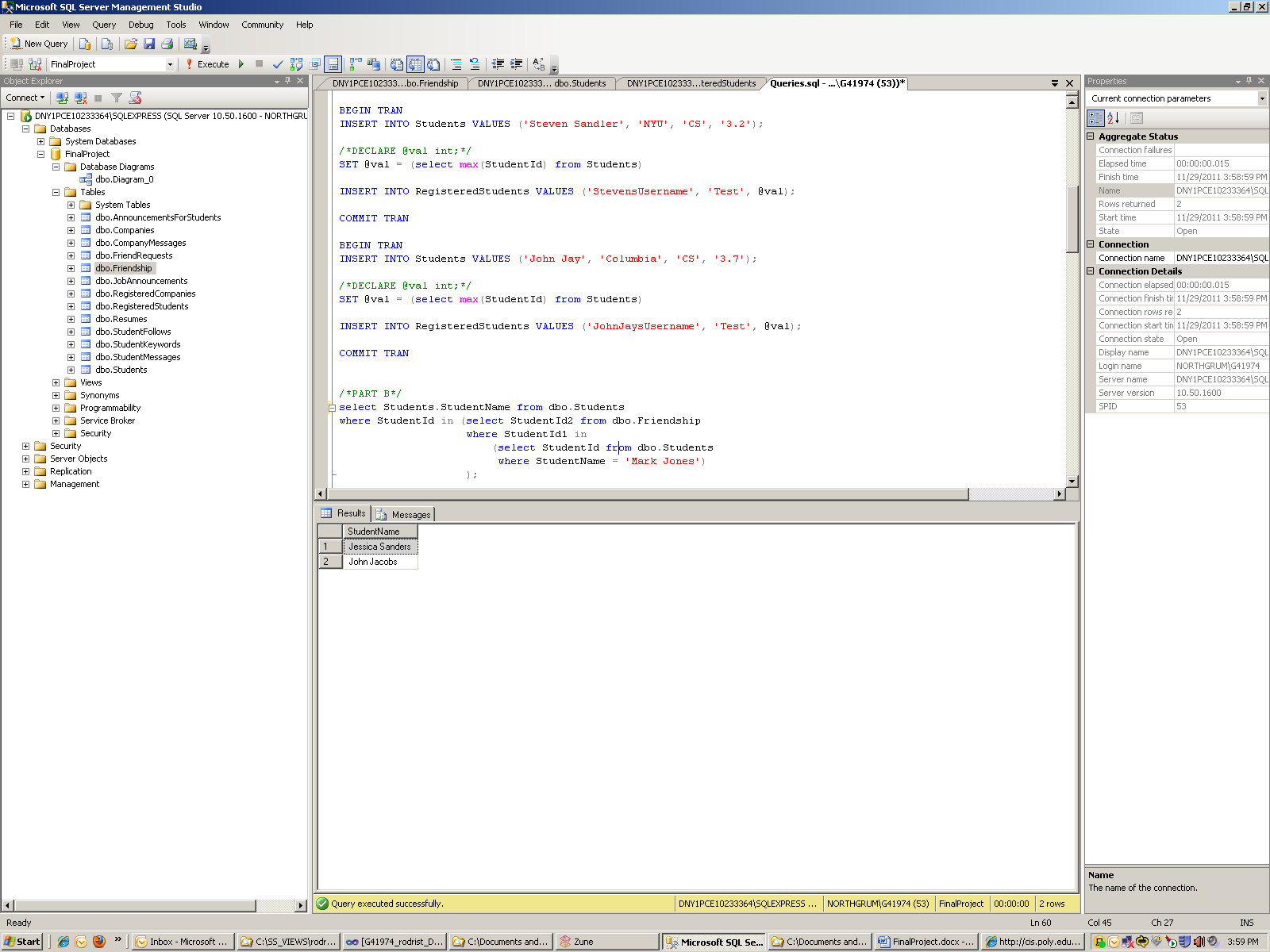
);

We can find a Student’s Id, then find all the student id’s of his friends and then lookup the names of those friends.

In the query we are looking for friends of Mark Jones so we need to create a listing of his friends in the friendship table. Mark is student ID 7, so he is friends with Jessica and John.



After we execute the query as expected we get,



(C)

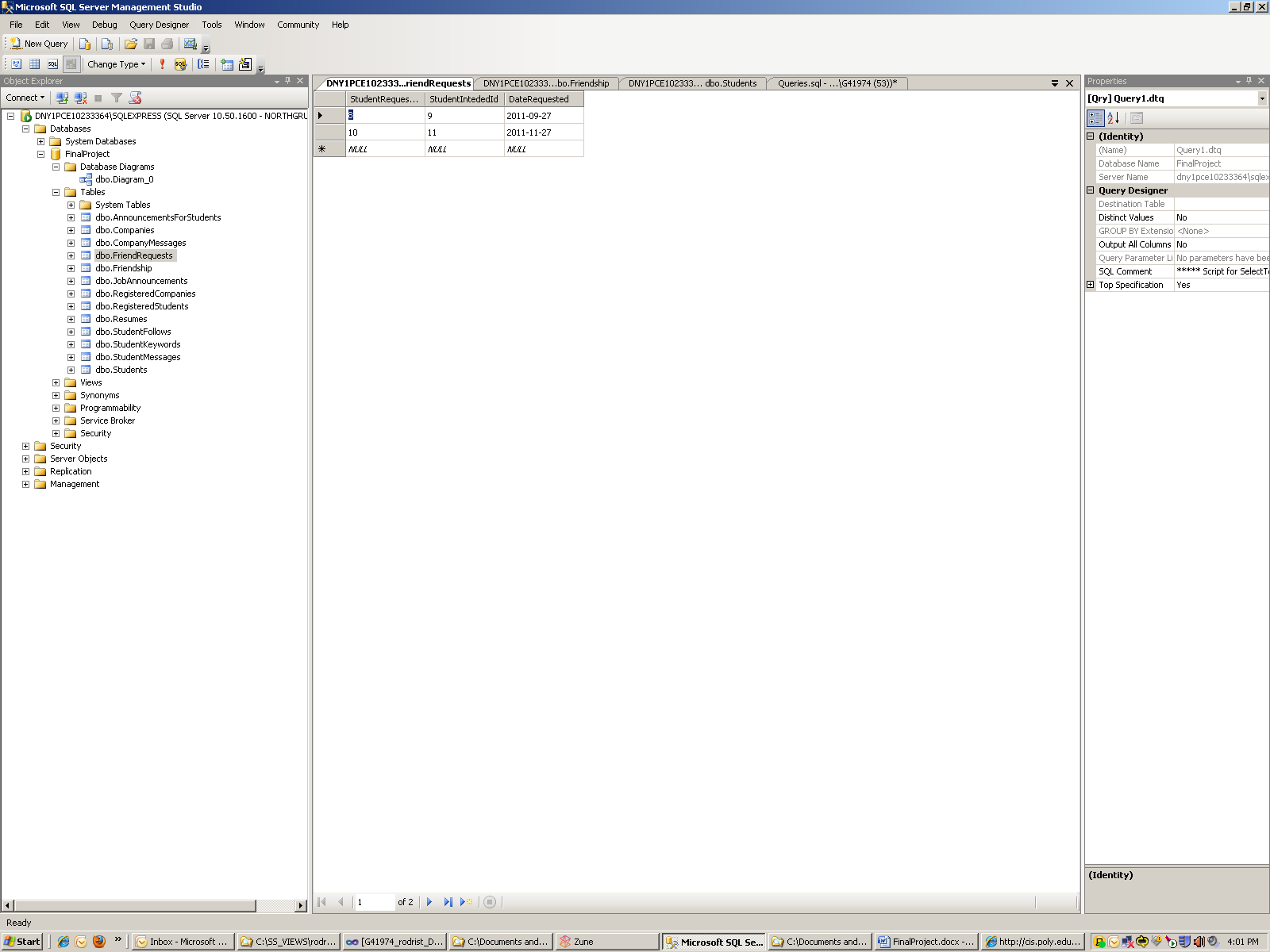
Delete all friendship requests that are older than one month and that have not been answered

DELETE FROM FriendRequests

WHERE DATEDIFF(Month, DateRequested, getDate()) > 1;

Friendship requests have their own table. Therefore any request over a month old can simply be removed.

We need to delete all friendship requests older than one month. For this example we create the following,



After the query we should be left with one entry.



(D)

List all students from NYU that are following Microsoft

select Students.StudentID, Students.StudentName from StudentFollows

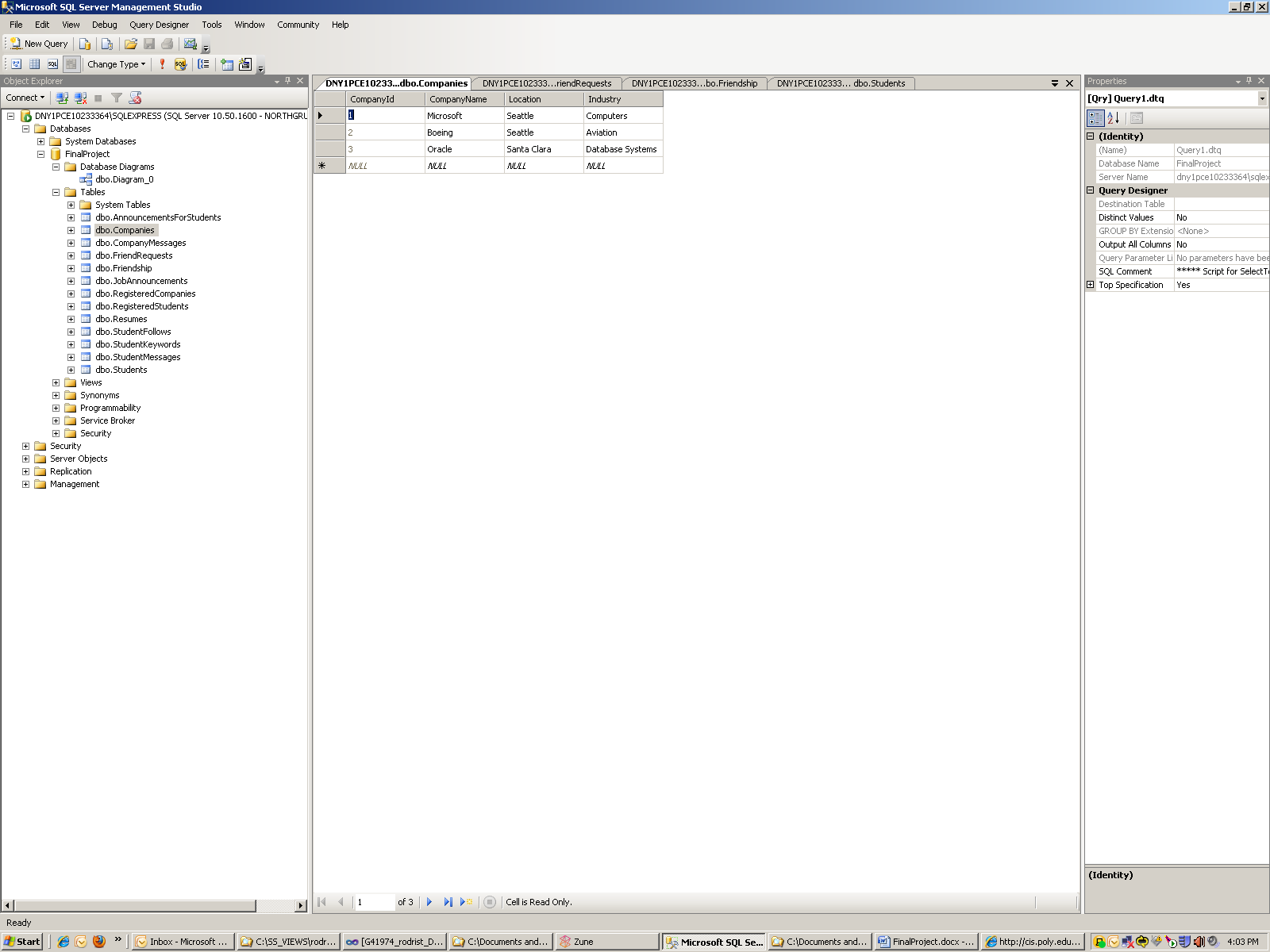
INNER JOIN Students on StudentFollows.StudentId = Students.StudentID

INNER JOIN Companies on StudentFollows.CompanyId = Companies.CompanyId

where Students.University = 'NYU' and Companies.CompanyName = 'Microsoft';

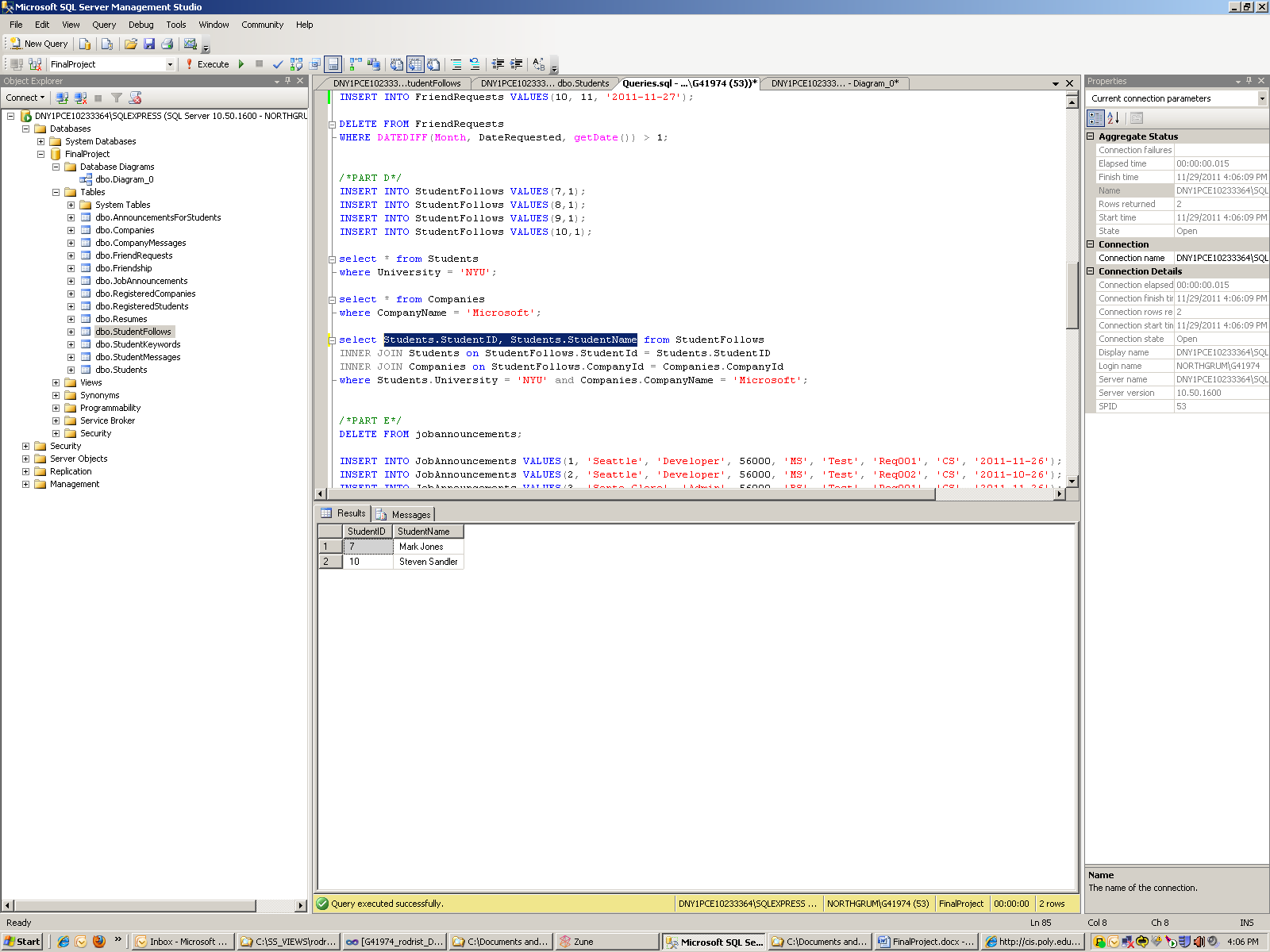
We can join the follows table with the students and companies table and then simply filter using a where clause by NYU and Microsoft.

In order to complete this query, we need to enter sample companies. This means we need an entry for Microsoft and something other than Microsoft, and then create entries for students following those companies.





Above we can see that there are 4 different students following Microsoft (Company ID 1). The question wants students from NYU. This would yield student ID 7 and student ID 10.



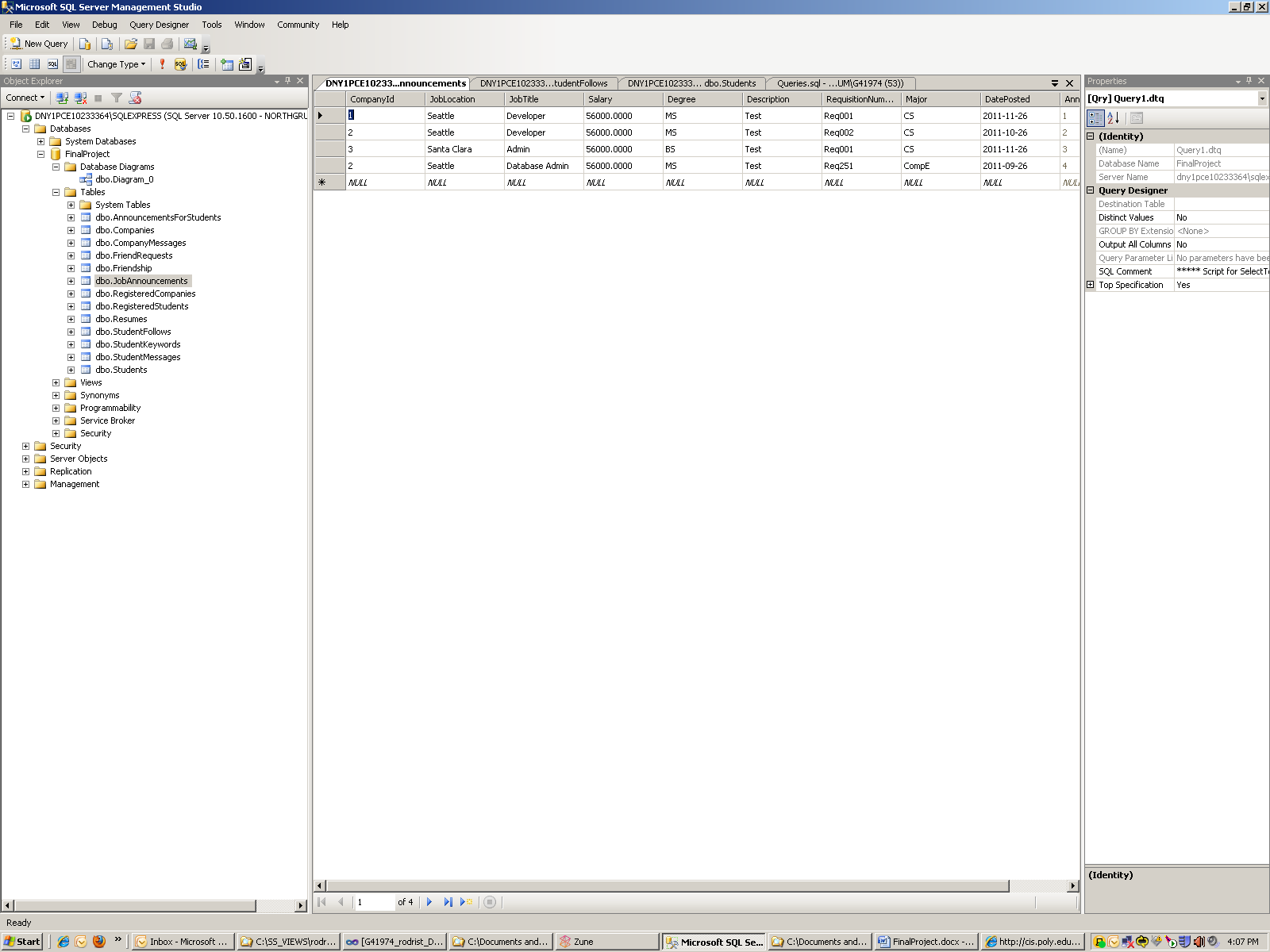
List all job announcements posted in the last week that are looking for someone with an MS in CS

select \* from jobannouncements

where Degree = 'MS' and Major = 'CS' and DATEDIFF(week, DatePosted, Getdate()) <= 1;

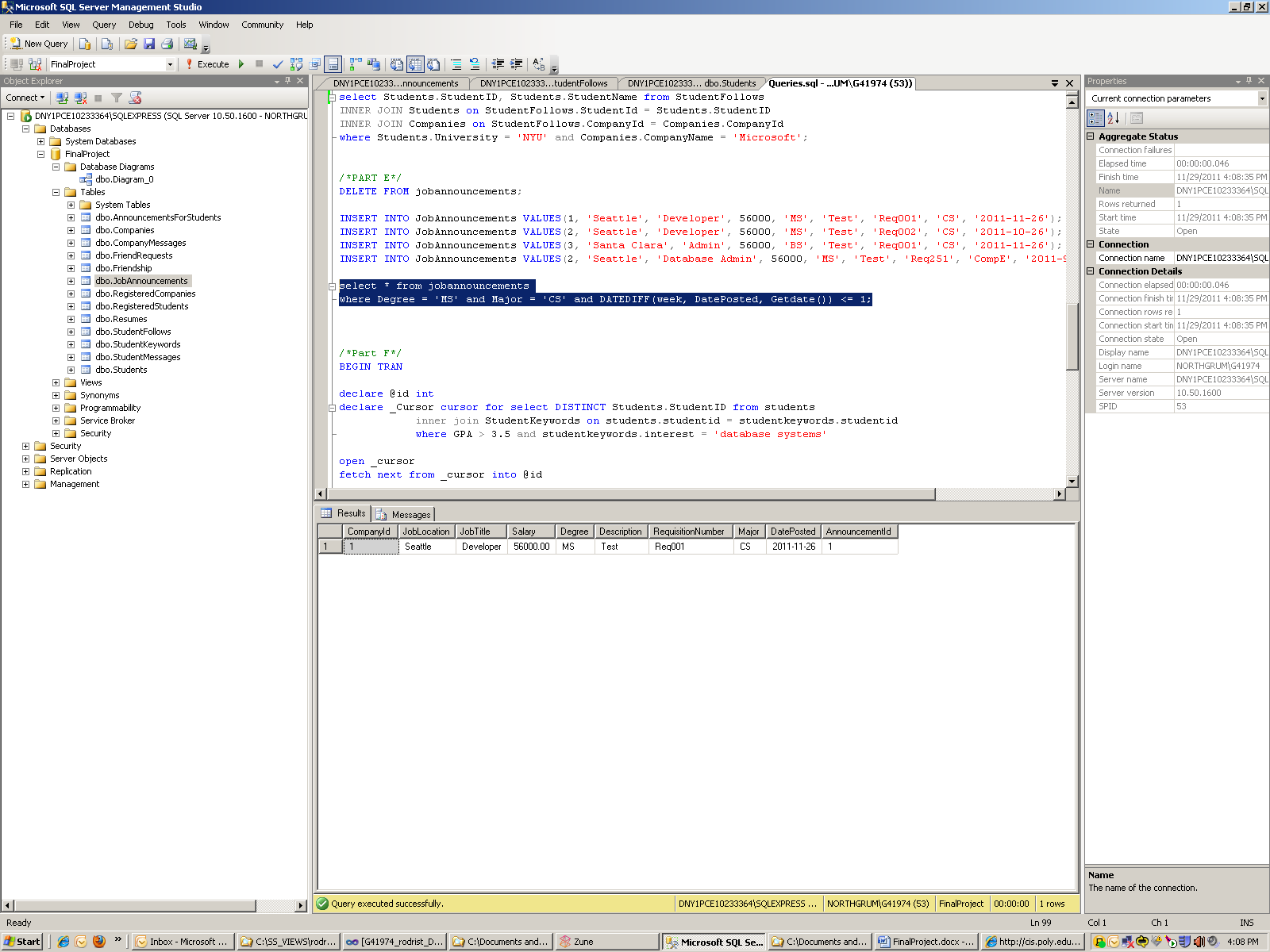
Simply query the jobannouncements table on the required info

We create the following job announcements:



We are looking for the ones on MS and CS. Those are rows 1 and 2. We are only interested in those in the last week which is row1.

After the query:



(F)For each student with GPA > 3.5 whose resume contains the keywords “database systems”, create a notification telling the student about a particular new job announcement that a company has posted

BEGIN TRAN

declare @id int

declare \_Cursor cursor for select DISTINCT Students.StudentID from students inner join StudentKeywords on students.studentid = studentkeywords.studentid where GPA > 3.5 and studentkeywords.interest = 'database systems'

open \_cursor

fetch next from \_cursor into @id

while @@fetch\_status = 0

begin

fetch next from \_cursor into @id

print @id

INSERT INTO AnnouncementsForStudents VALUES (@id, 2)

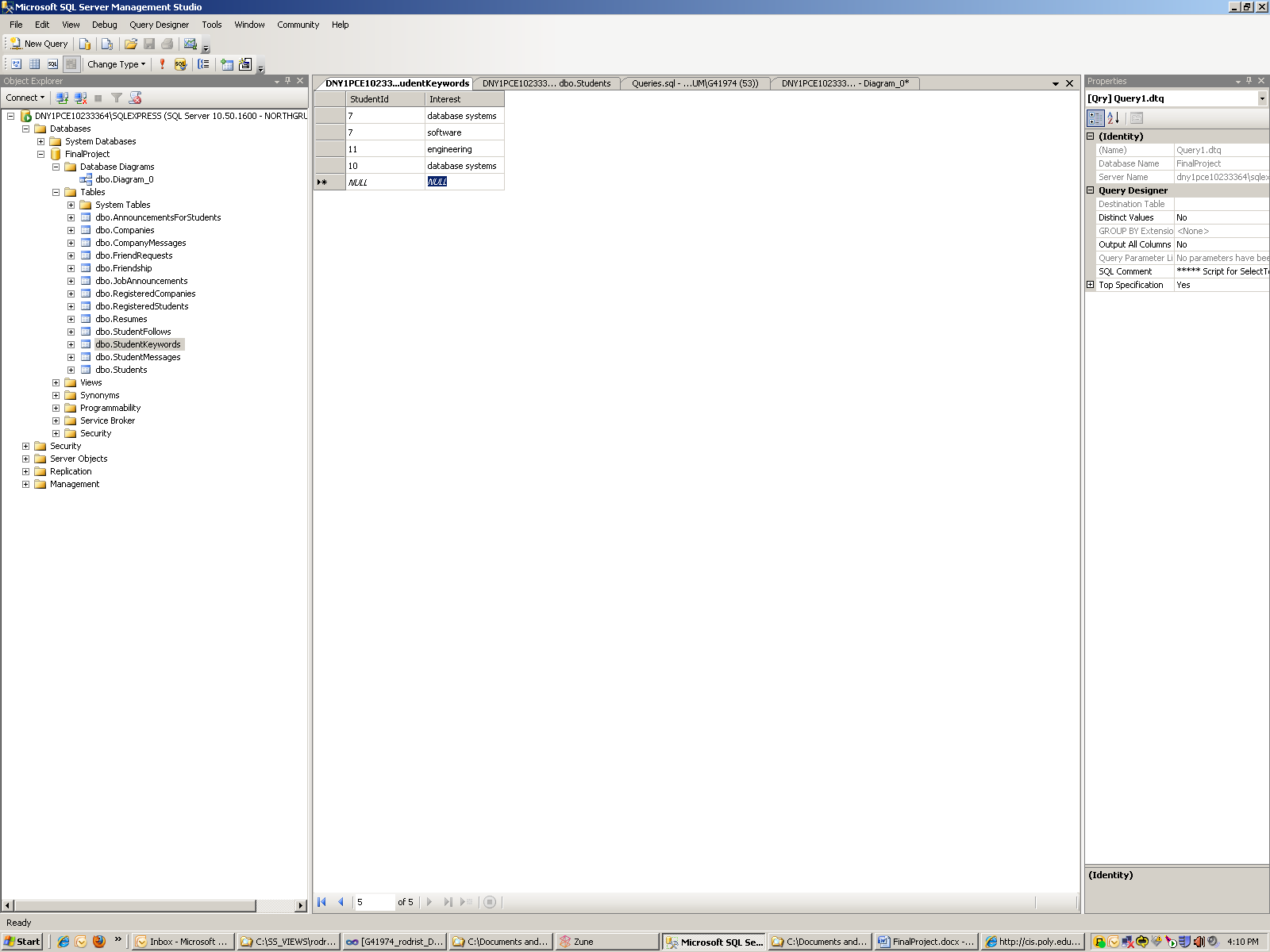
End

close \_cursor

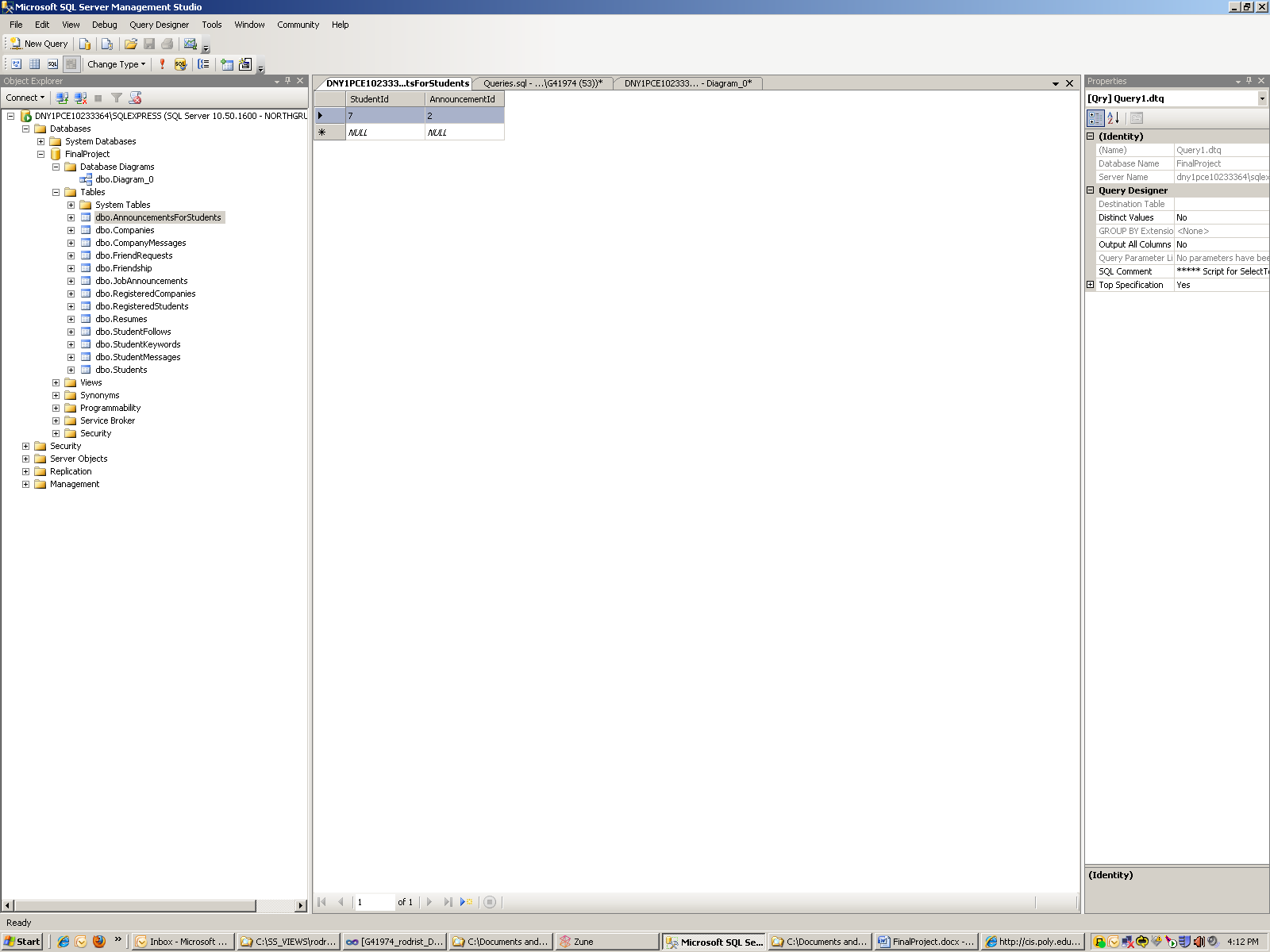
deallocate \_cursor

COMMIT TRAN

We already created two students with GPA > 3.5, Mark(7) and John(11). We now need to create keywords for these students. We also add another student Steven(10) who does not have a 3.5 GPA but has an interest in database systems. This should be filtered out on the query.



Mark should be the only valid result. As part of the query we just randomly choose to notify any of the students about JobAnnouncementID of 2.



We can see that only Mark has the announcement of the job position as the result of the query

**Stored Procedures – These are procedures created to help process certain types of transactions. We could create stored procedures for every type of sql transaction the front end makes. But this would not entirely be necessary. Most of the sql work can be done in the php code.**

Registration –

CREATE PROCEDURE [dbo].[RegisterStudent]

@username varchar(50),

@password varchar(50),

@name varchar(50),

@university varchar(50),

@major varchar(50),

@gpa float

AS

INSERT INTO Students VALUES (@name, @university, @major, @gpa);

DECLARE @val int;

SET @val = (select max(StudentId) from Students)

INSERT INTO RegisteredStudents VALUES (@username, @password, @val);

Print @Val;

RETURN @val;

We have an identical procedure for registercompany. This procedure wil be called by the front end on the registration page of the site. A student will populate a form with his desired username, password and personal information. This will enter the information into the required databases.

Get Profile Information

CREATE Procedure [dbo].[GetCompanyProfile]

@id int,

@username varchar(50) OUTPUT,

@password varchar(50) OUTPUT,

@name varchar(50) OUTPUT,

@location varchar(50) OUTPUT,

@industry varchar(50) OUTPUT

AS

select @name = CompanyName, @location = Location, @industry = industry from companies where CompanyId = @id;

select @username = Username, @password = Password from RegisteredCompanies where CompanyId = @id;

We have a similar procedure for the students. This will allow a student to see his profile and edit it if he needs to.

Update Profile Information

CREATE procedure [dbo].[updatecompanyprofile]

@id int,

@username varchar(50) OUTPUT,

@password varchar(50) OUTPUT,

@name varchar(50) OUTPUT,

@location varchar(50) OUTPUT,

@industry varchar(50) OUTPUT

as

update companies

SET CompanyName = @name, Location = @location, Industry = @industry

where companyid = @id;

update registeredcompanies

SET Username = @username, Password = @password

where companyid = @id;

This procedure works hand in hand with the previous one. It allows a student or company to update their personal information.

Create announcement

CREATE procedure [dbo].[createnewannouncement]

@id int,

@jobtitle varchar(50),

@joblocation varchar(50),

@salary money,

@degree varchar(50),

@major varchar(50),

@reqnumber varchar(50),

@description varchar(max),

@date datetime

as

INSERT INTO [FinalProject].[dbo].[JobAnnouncements]

([CompanyId]

,[JobLocation]

,[JobTitle]

,[Salary]

,[Degree]

,[Description]

,[RequisitionNumber]

,[Major]

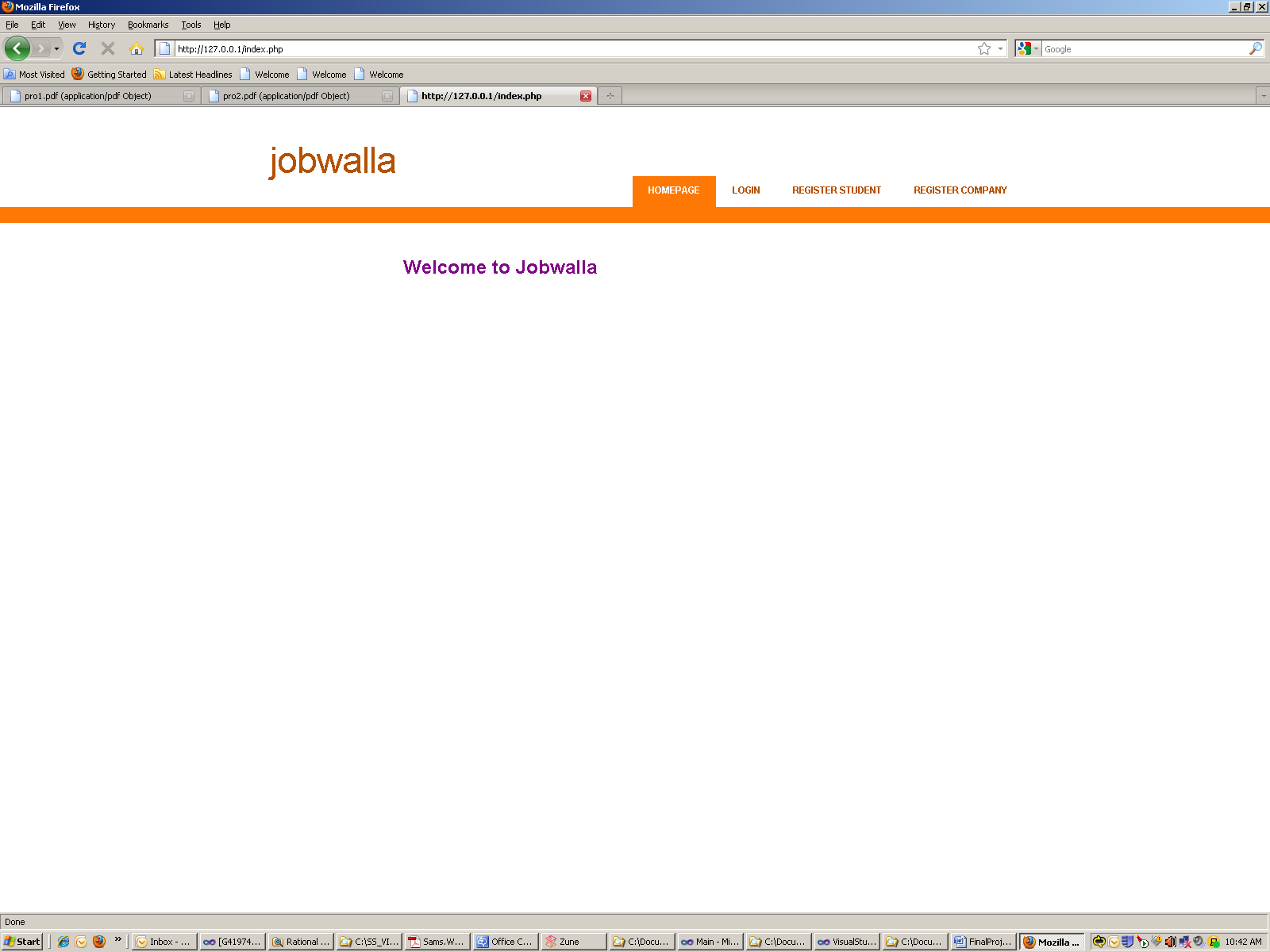
,[DatePosted])

VALUES

(@id, @joblocation, @jobtitle, @salary, @degree, @description, @reqnumber, @major,@date);

Finally this procedure allows a company to fill out a form for a new announcement and we can enter into the correct table

**Jobwalla PHP Front End**

****

The front end to the site was written entirely in php. A css style sheet was obtained and used to provide a nice look and feel to the site. The structure of the site is very simple. Users have the ability to login or register. When they login, they are presented with their control panel on the left which provides them all the features available to them in the introduction.

We interact with the database using the php\_mssql.dll for Sql Server. This allows to both execute queries and stored procedures to the database. A single dbconnect.php file is imported by all pages to allow connected to the database. It contains the connection to the db as such

$link = mssql\_connect($server, username, password);

mssql\_select\_db('FinalProject', $link) or die('A error occured: ' . mysql\_error());

We can then query as follows:

$SQL = "SELECT studentid FROM RegisteredStudents WHERE Username='$uname' and Password='$pword'";

$result = mssql\_query($SQL);

$row = mssql\_fetch\_array($result);

Or

$proc = mssql\_init('updatecompanyprofile',$link);

mssql\_bind($proc,'@id',$id, SQLINT4);

mssql\_bind($proc,'@username',$username,SQLVARCHAR);

mssql\_bind($proc,'@password',$password,SQLVARCHAR);

mssql\_bind($proc,'@name',$name,SQLVARCHAR);

mssql\_bind($proc,'@location',$location,SQLVARCHAR);

mssql\_bind($proc,'@industry',$industry,SQLVARCHAR);

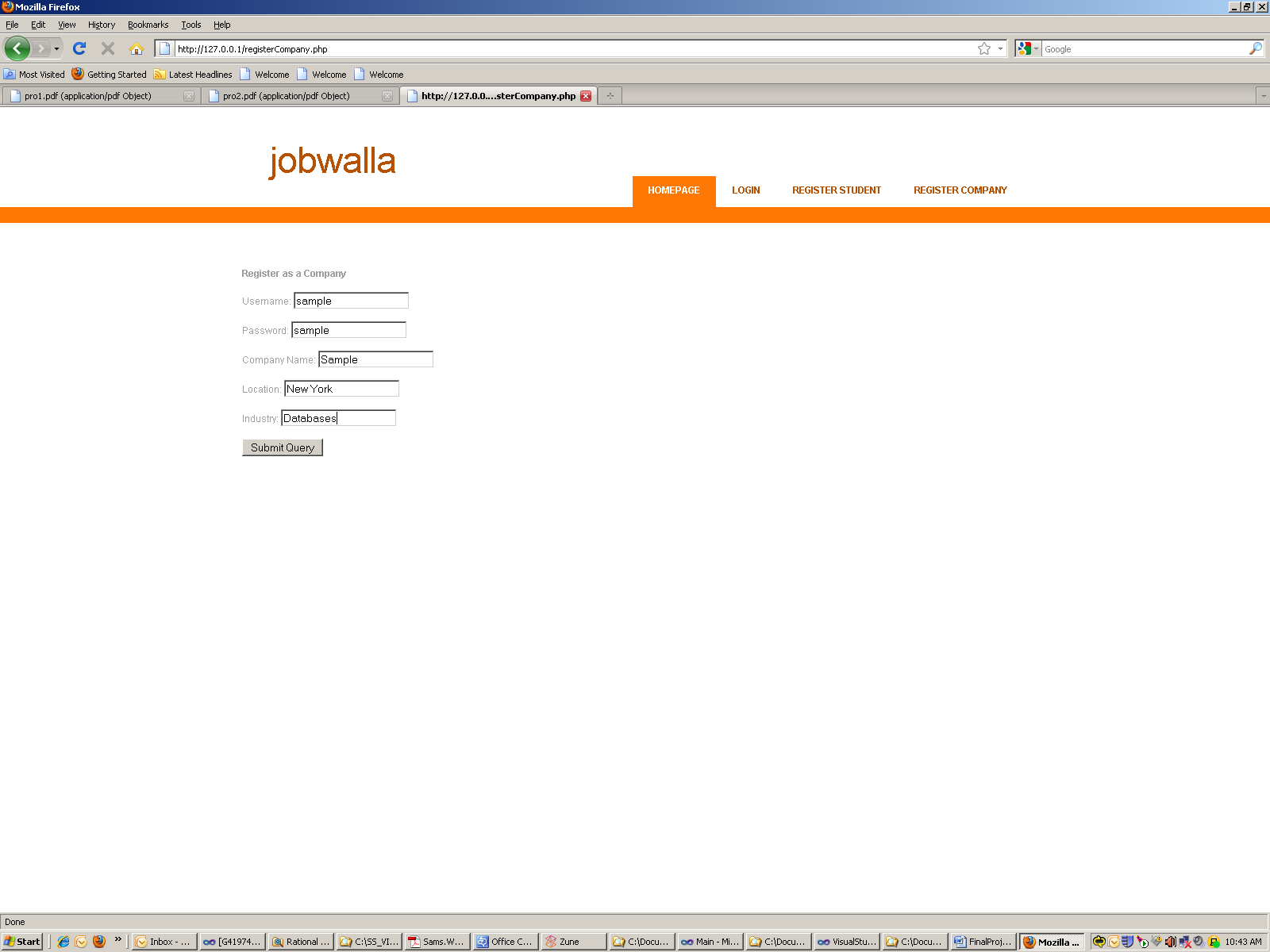
$result = mssql\_execute($proc);

In this way, it is extremely simple to create the php pages and query the db using either method.

**How to use JobWalla –**

**Company**

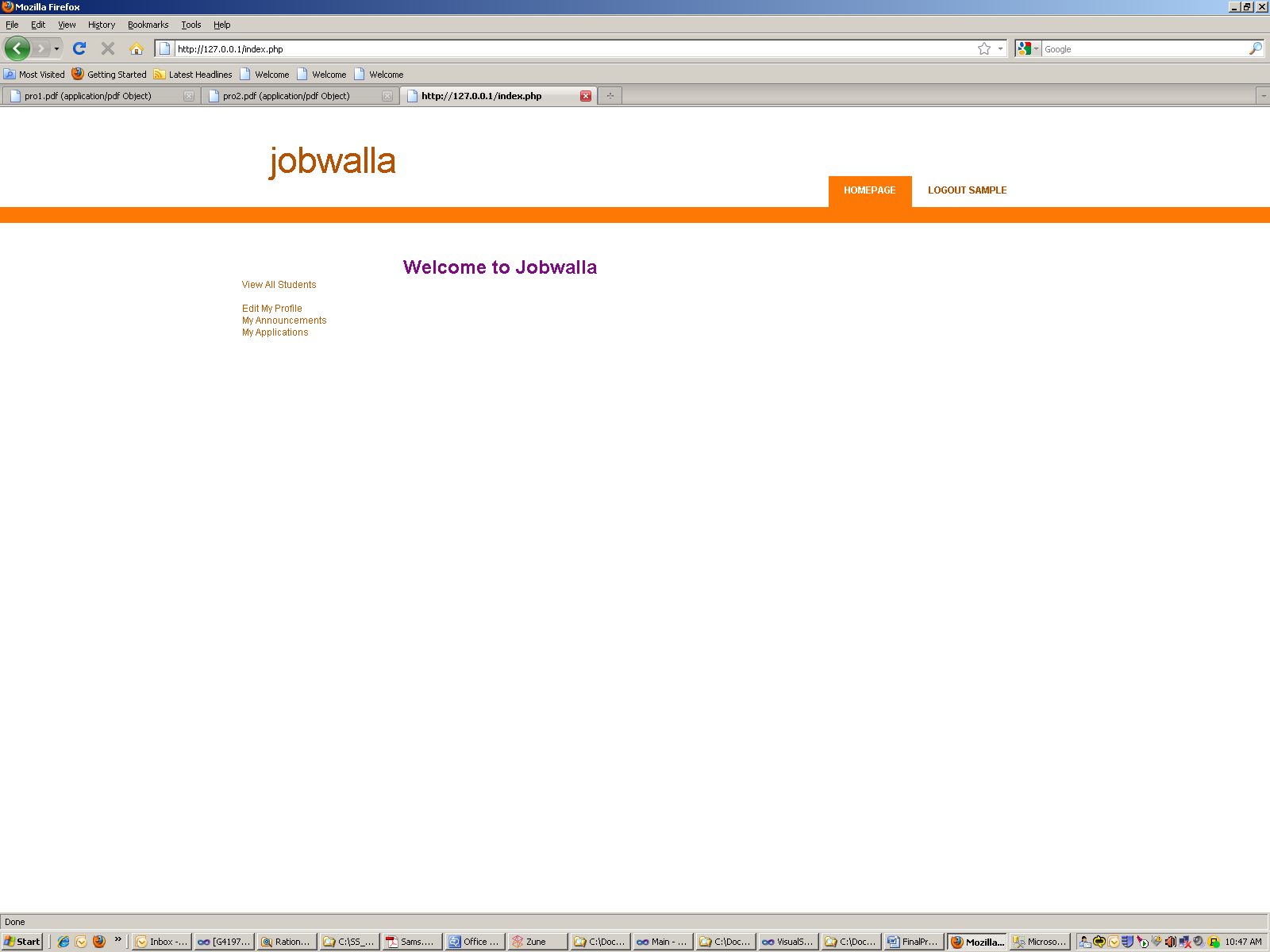
As a company user, the first thing you want to do is register on the main page by clicking the register company tab. We will register a company called. Sample.



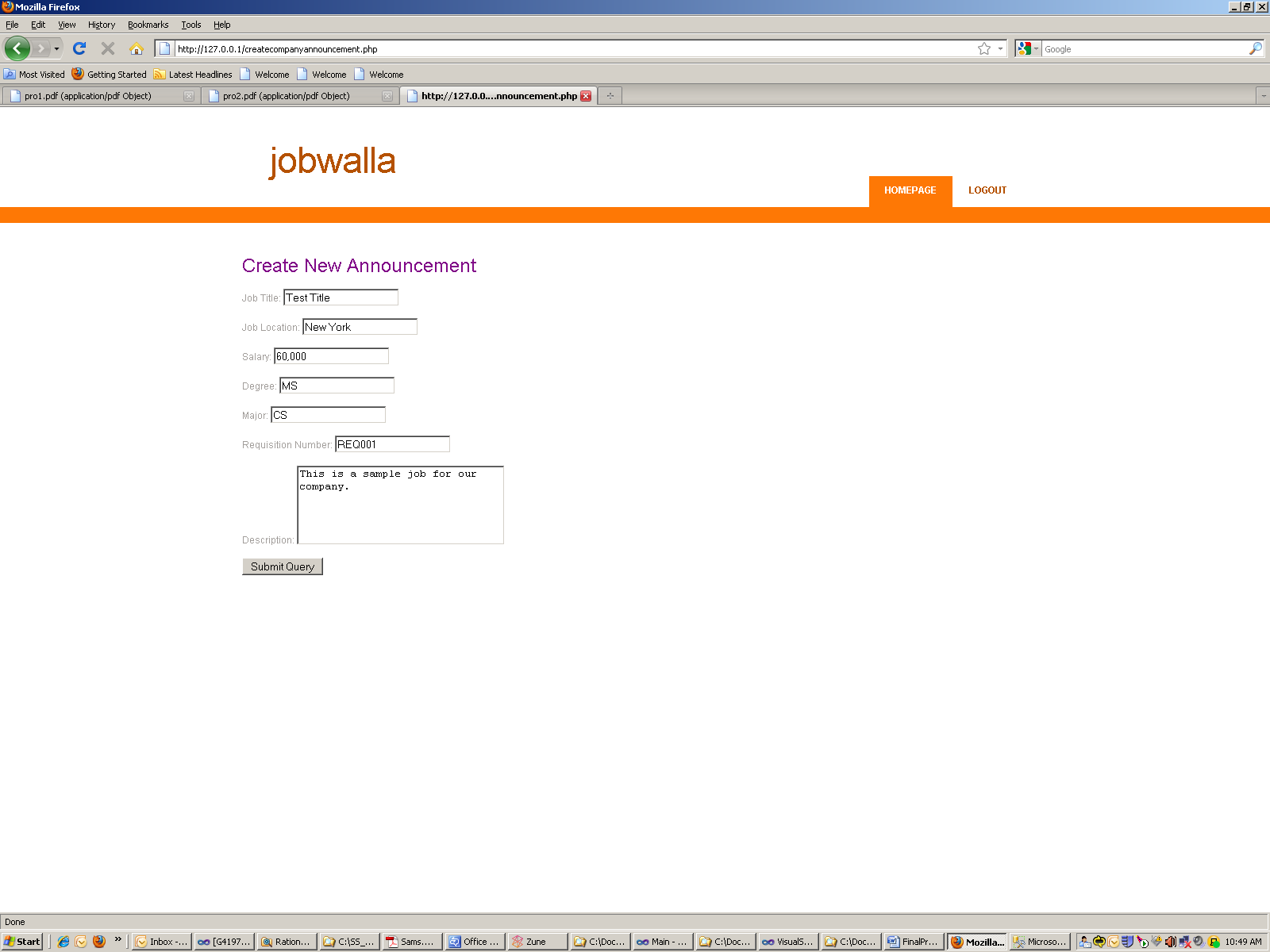
We can then login.



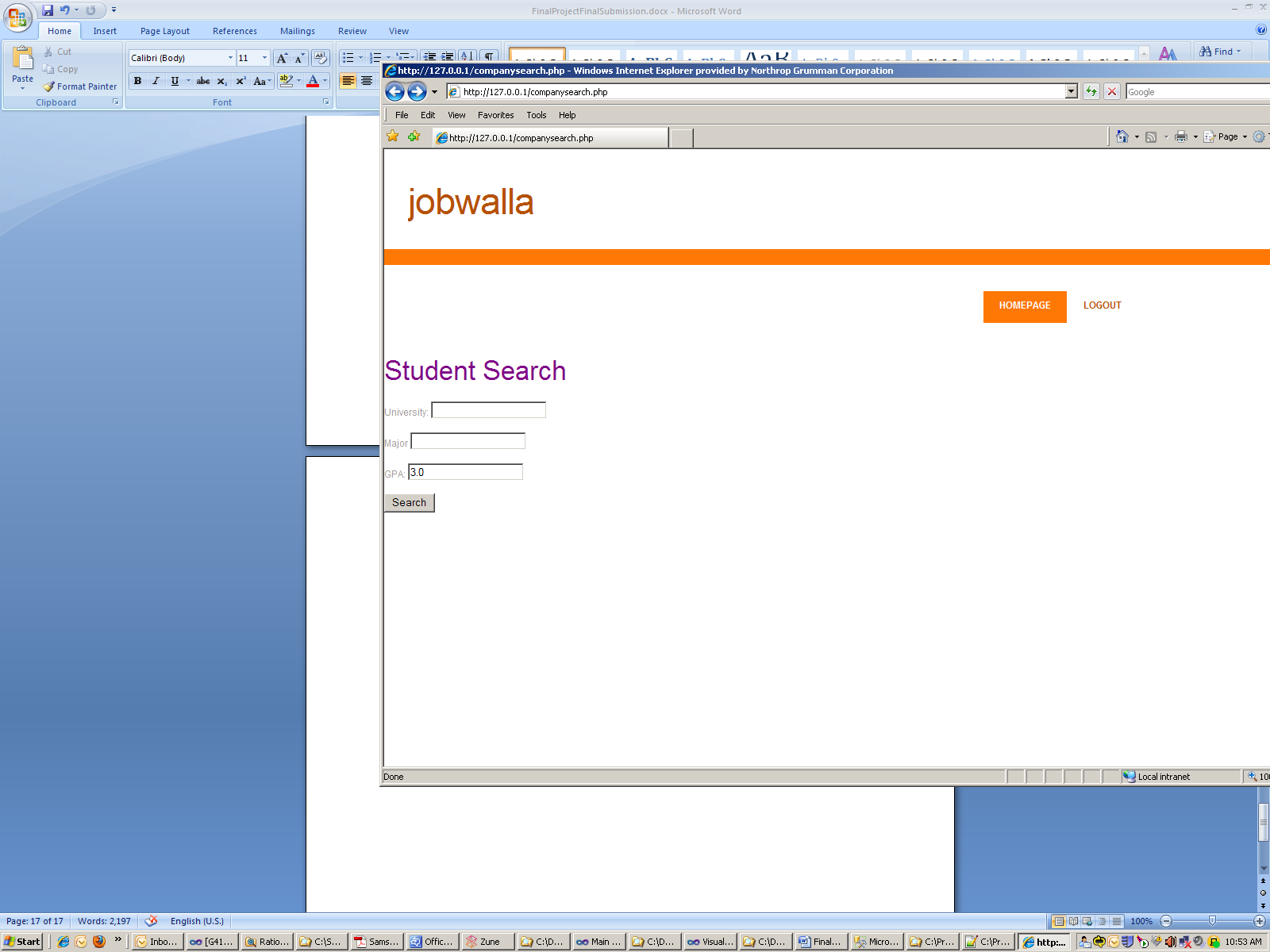
After we login we are presented with our company home page panel. Here we can view all students, edit our profile, look at our announcements and create new announcements, and look at all the applications for our announcements.



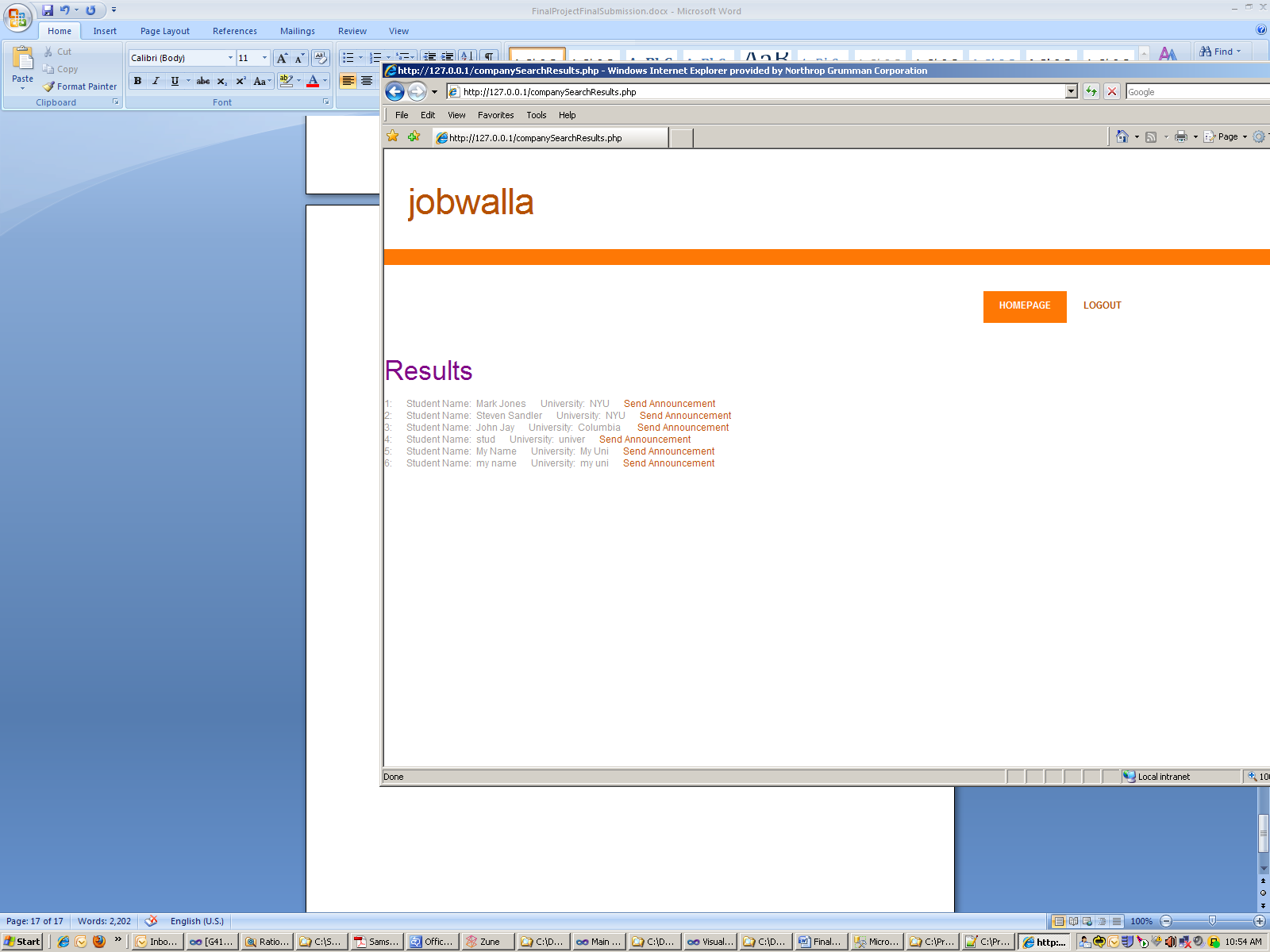
Lets create an announcement.



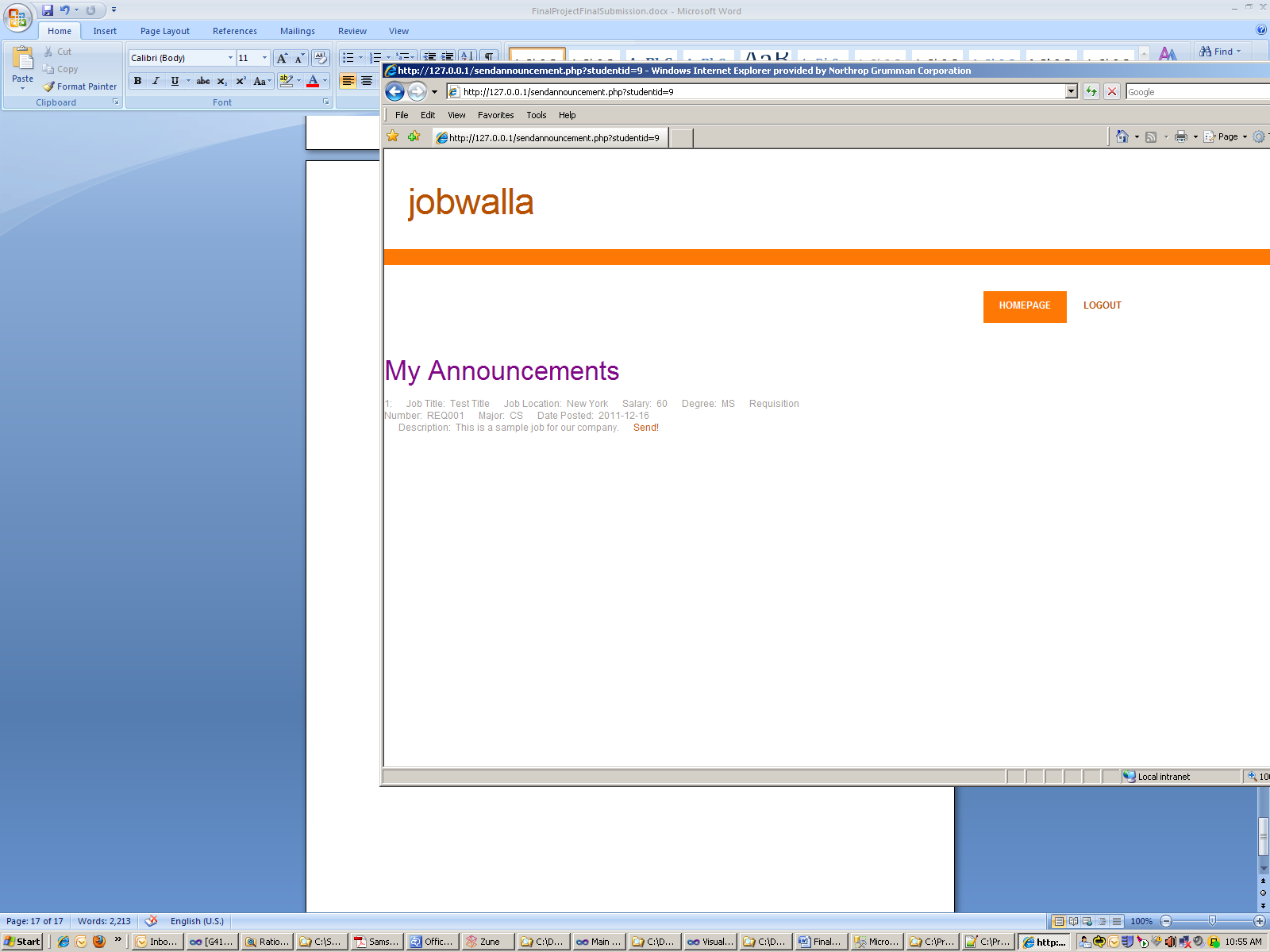
Now we can search for students whose GPA is above 3.0. By default, we have made the query search >= 3.0 as that is normally what companies what on a GPA search.



We are given the results:



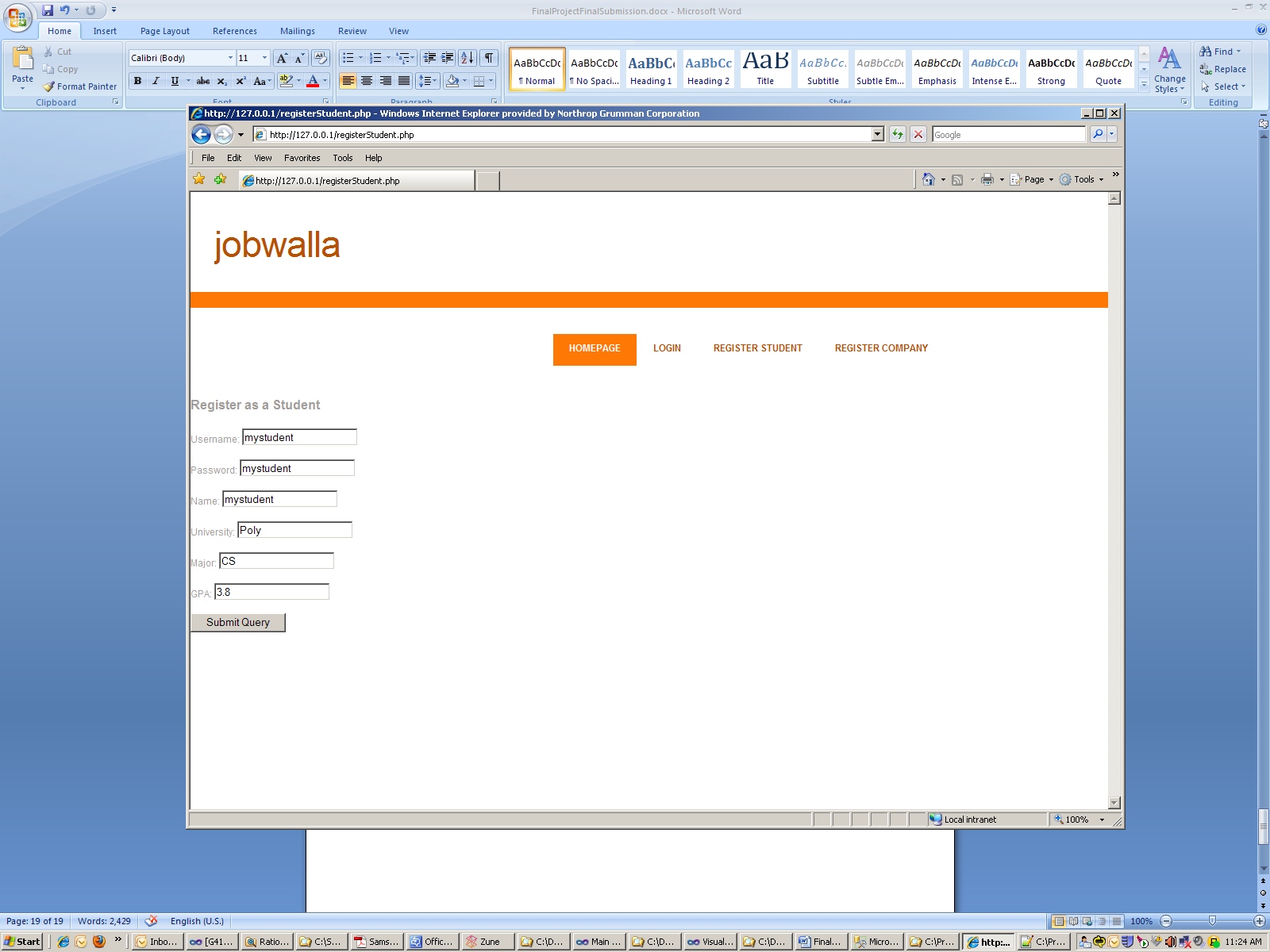
We can then easily send them the announcement we created earlier.



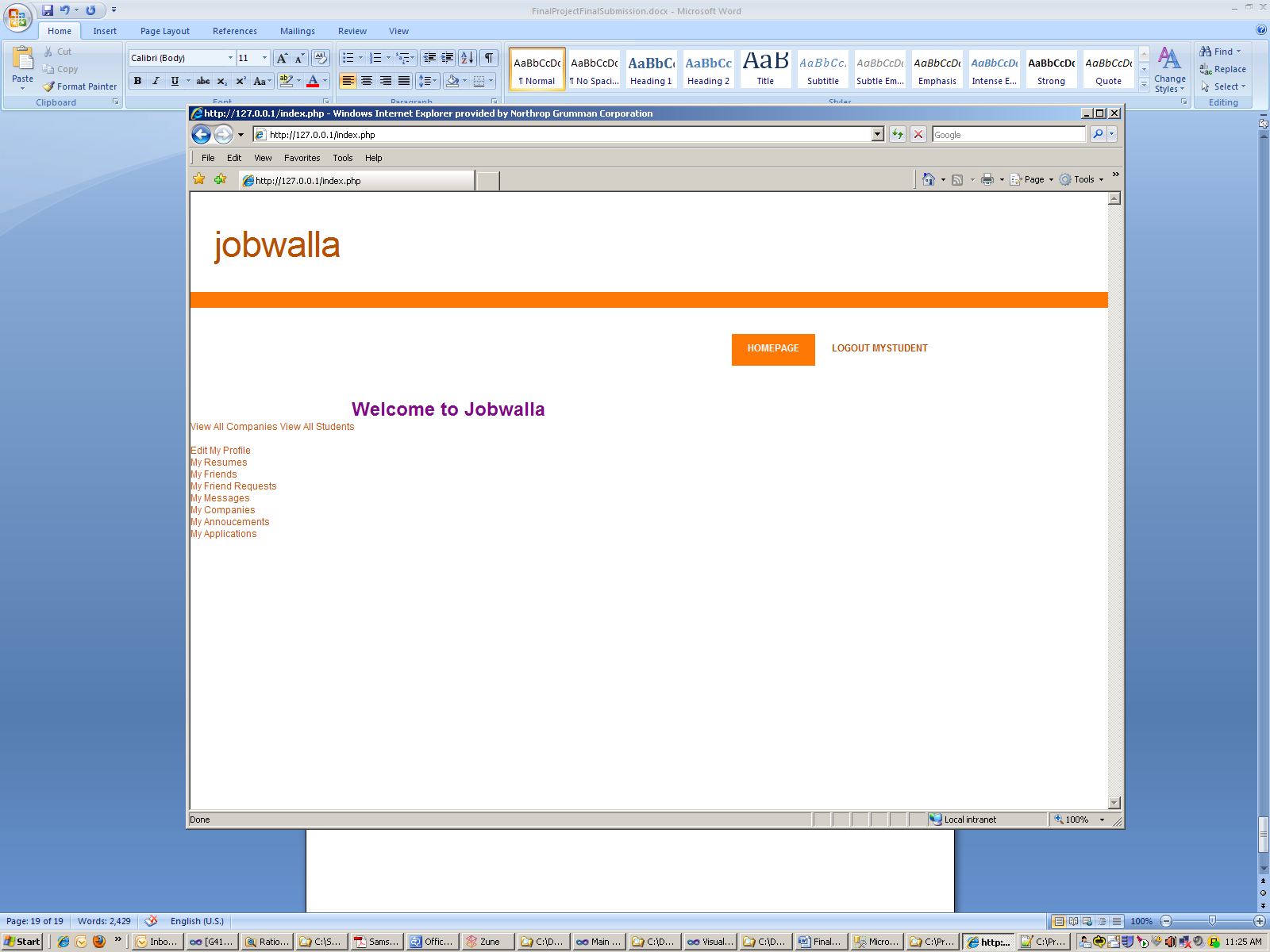
These are only some of the features shown. As you can see, it is extremely easy and straight forward for our users.

**Student**

Similary, a student must first register.

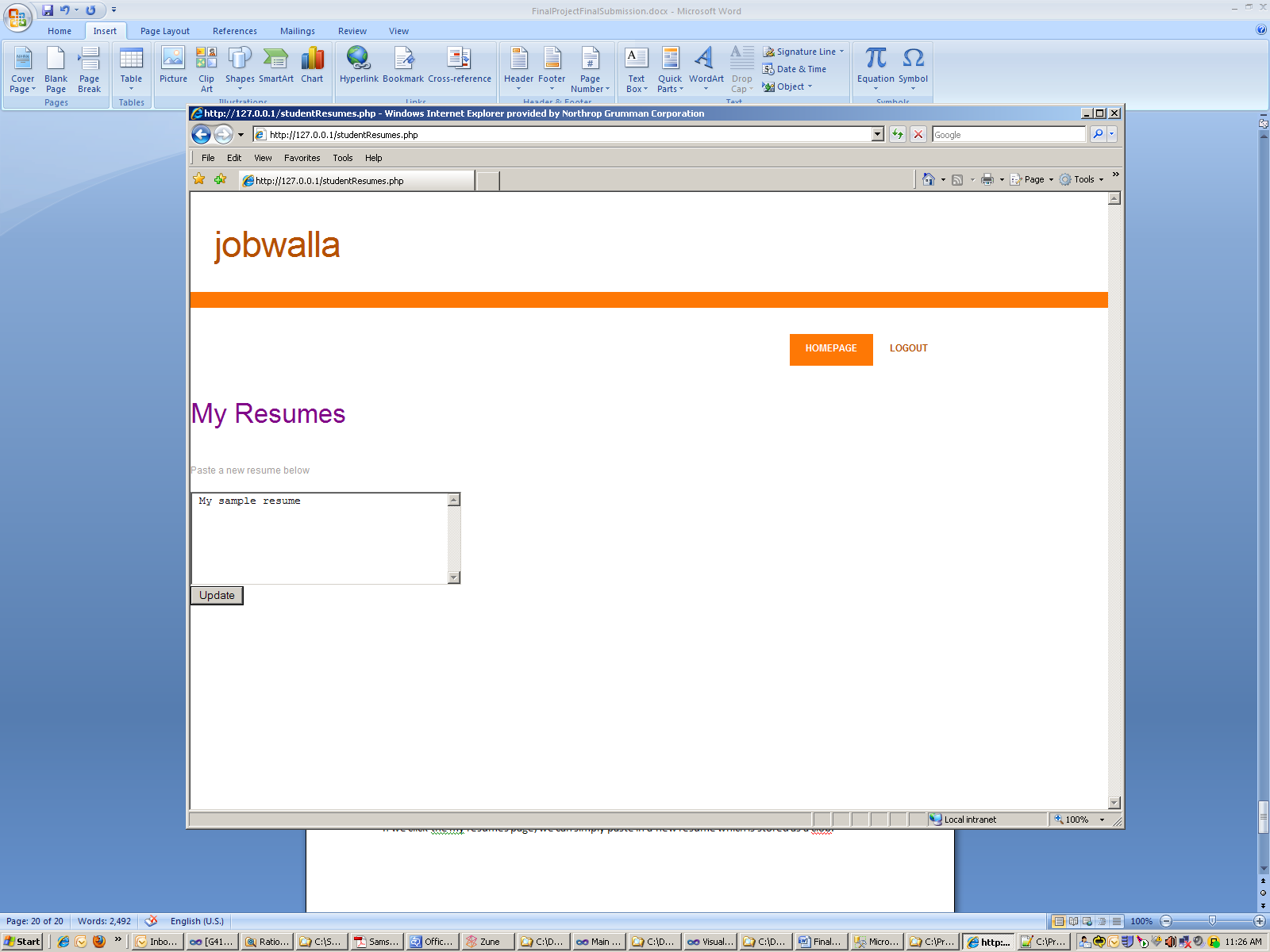


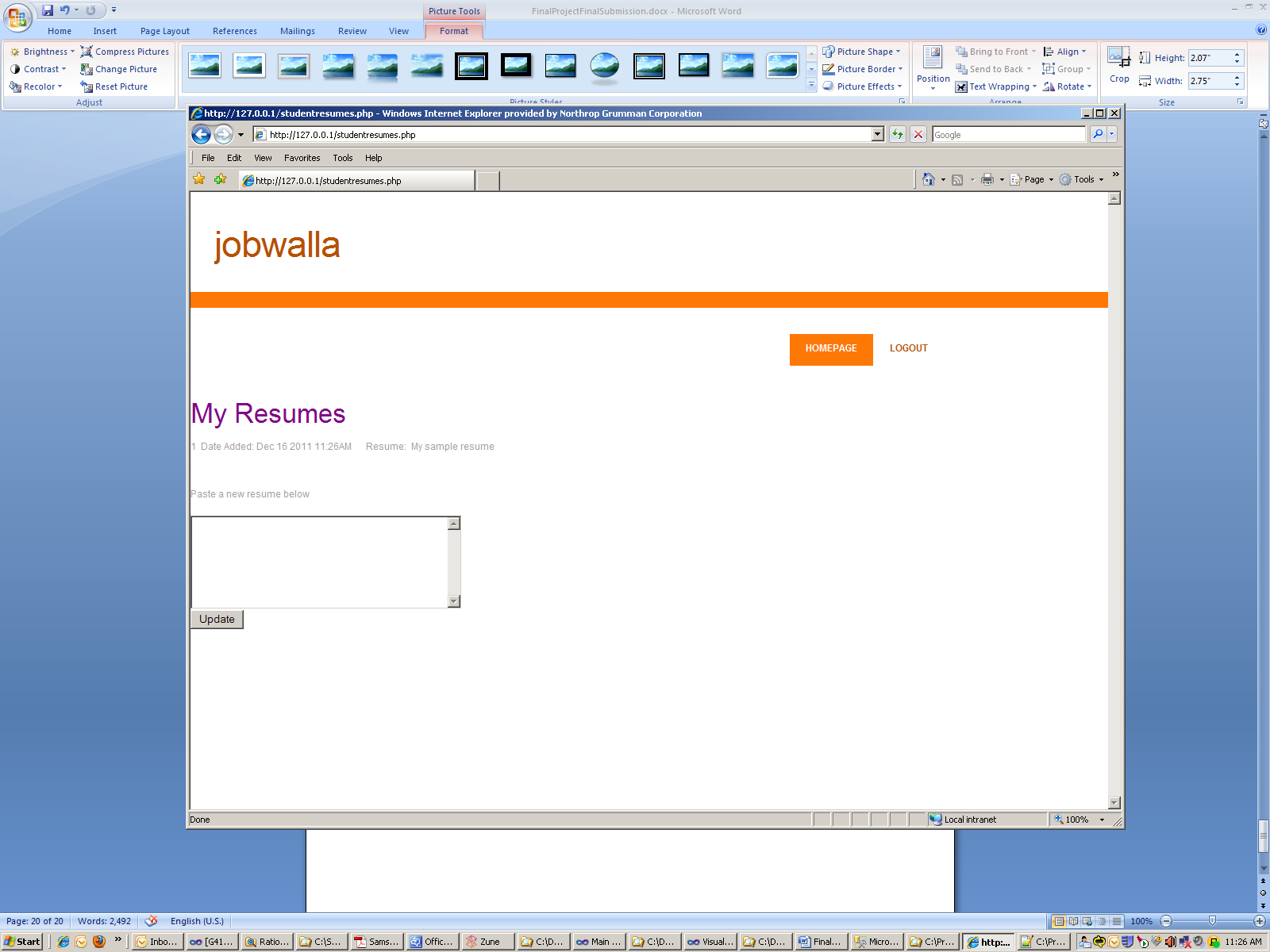
After logging in as shown before, we obtain our main control panel.



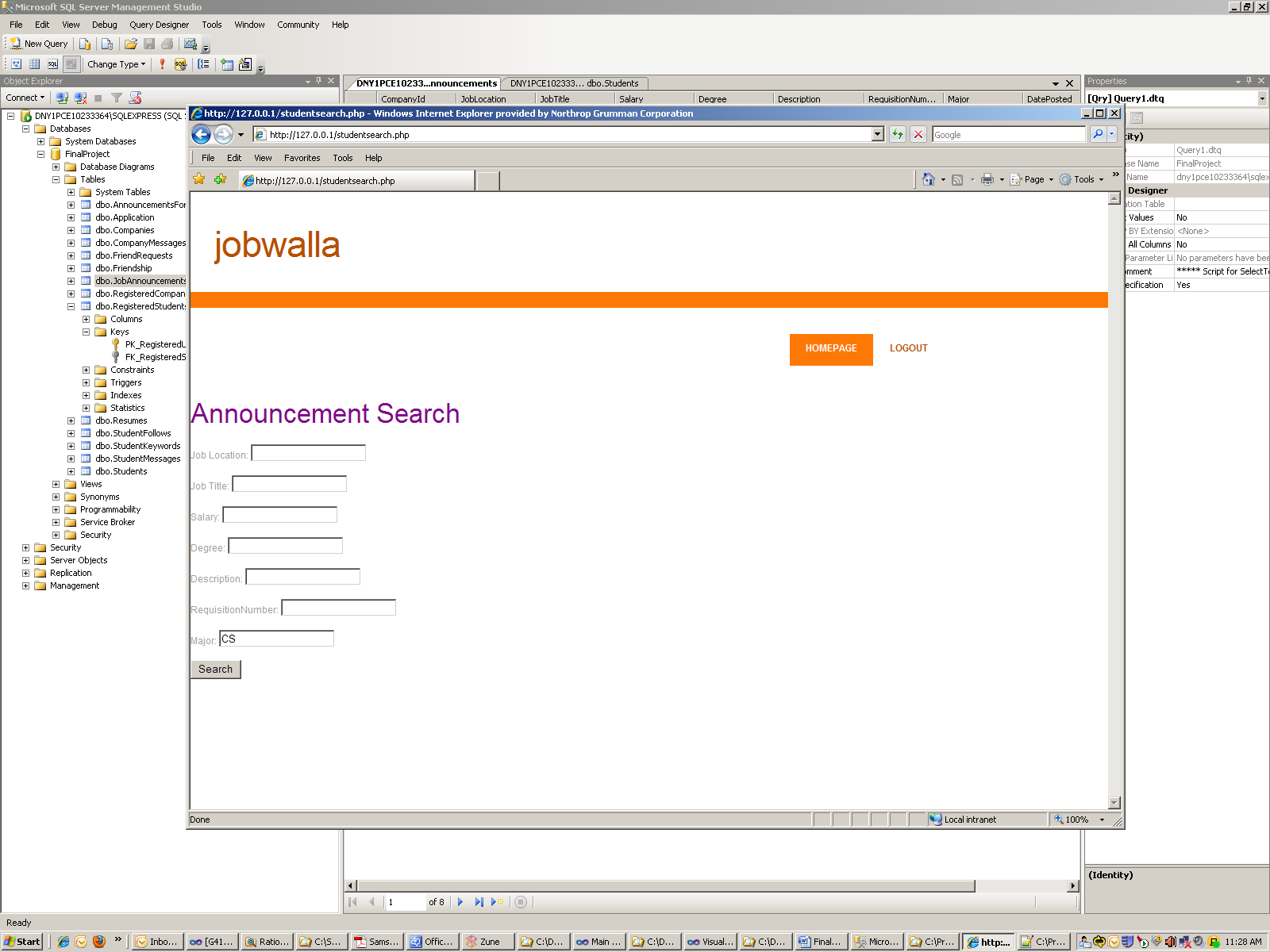
As you can see, all the features are immediately available to the student. We will show an example of how to create a resume and apply for a job.

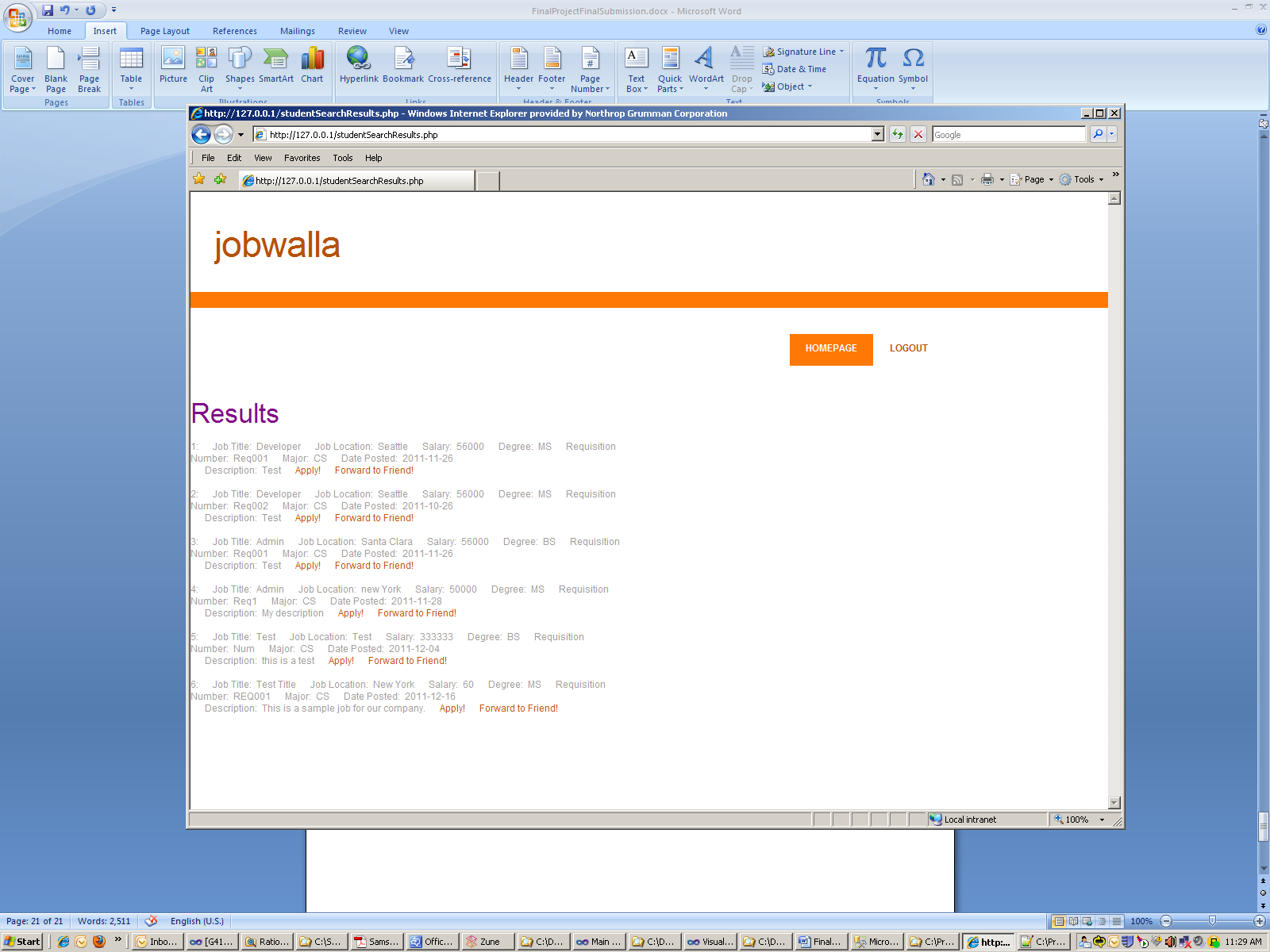
If we click the my resumes page, we can simply paste in a new resume which is stored as a clob.





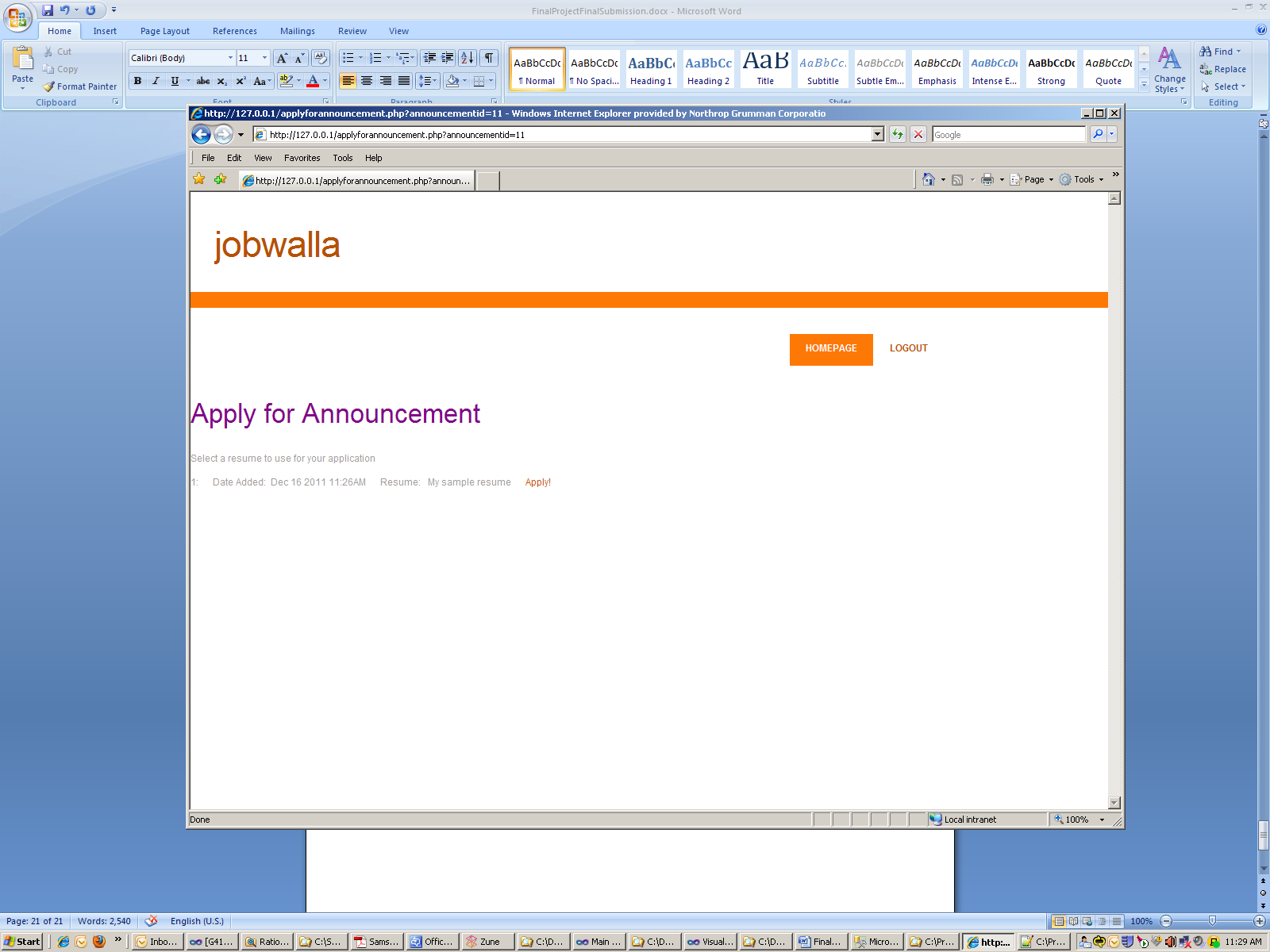
Now we can go back to the homepage and search for announcements. We search for all CS jobs.





We can easily choose to forward this to a friend, or apply for it ourselves. We will apply for the job position we created earlier as a sample company.

We choose which resume we wish to use and apply!



Now the company can simply see our resume and choose to contact us.

**Conclusion**

The Jobwalla site is a simple and easy to use networking site which has many features available to students and companies. They can easily search for required criteria and apply or choose to contact students for job opportunities. Students easily friend each other and message each other as they choose.

The backend database, easily accomplishes all of our goals and makes it easy for the front end to focus primarily on user experience only querying the database for information. Overall our site is complete and provides full functionality to the user.