

Andrey Grzegorzewski  
 Programming Environments  
 Ohio Northern University – Fall 2017  
 Progress Report 2


## Introduction and Status



This is the second progress report covering my app, JobFairsApp, which will perform data management and interview scheduling for job fairs. Since the last progress report, I have finished all of the data entry forms. All data entry can be done through JobFairsApp and verified in SSMS. I still need to sanitize my inputs in the newer forms; for example, it is possible to break some forms by attempting to input the same data twice. Currently, it is not possible to edit or delete data from the app. I have started working on my algorithm for scheduling interviews, and have made significant progress toward implementing the complete algorithm as discussed in class.

## Updates to the Project

### *Database Design Updates*

I have only made two updates to the database since my last progress report. First, I updated the Interviews table to reflect more specific information about the interview. I left the already-existing columns unaltered, but added columns for CompanyID, EmploymentTypeID, and SpecialtyID. They are foreign keys to the ID columns in the Companies, EmploymentTypes, and Specialties tables, respectively.

	Column Name	Data Type	Allow Nulls
	ID	int	<input type="checkbox"/>
	JobFairID	int	<input type="checkbox"/>
	CandidateID	int	<input type="checkbox"/>
	TimeSlotID	int	<input type="checkbox"/>
	TableID	int	<input type="checkbox"/>
	CompanyID	int	<input checked="" type="checkbox"/>
	EmploymentTypeID	int	<input checked="" type="checkbox"/>
	SpecialtyID	int	<input checked="" type="checkbox"/>

	Column Name	Data Type	Allow Nulls
	InterviewID	int	<input type="checkbox"/>
	InterviewerID	int	<input type="checkbox"/>

I also added an InterviewInterviewers table. This allows each interview to have more than one interviewer. This table has an InterviewID column (which is a foreign key

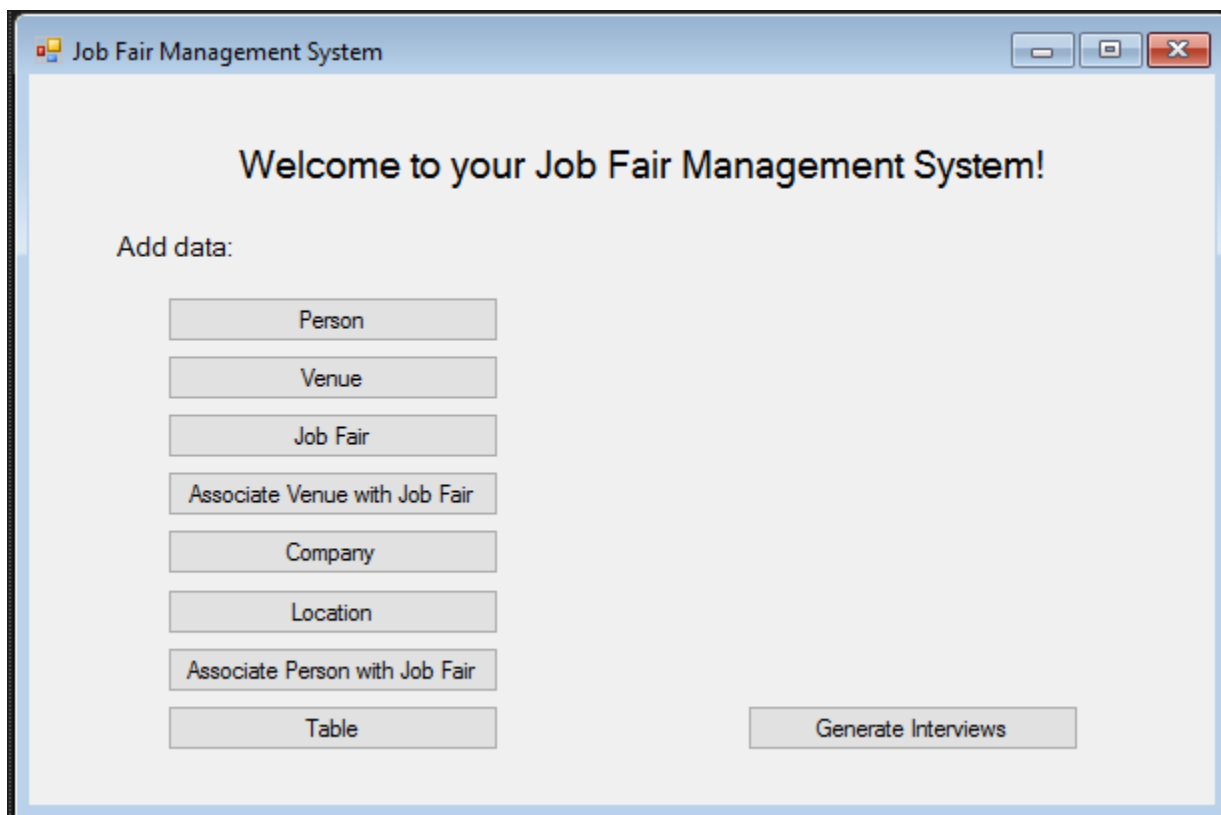
to the Interview table's ID column) and an InterviewerID column (which is a foreign key to the People table's ID column). The foreign key relationships will be used to join the Interviews and InterviewInterviewers table together to create a report including all of the

information from the Interviews table along with each interviewer who will be present at each interview.

### *App Design and Implementation Updates*

Since my last progress report, I have finished creating all of the forms necessary for data entry. This includes forms for Interviewers, Candidates, Locations, Tables, associating a person with his specialties, and associating a person with a job fair. These forms follow the same design pattern that has applied to the other forms so far.

Screenshots and descriptions of the updated main form, as well as the new forms, are shown below.



Since my last progress report, I've added buttons on the main form for adding locations and tables, as well as associating a person with a job fair. The rest of the new forms are shown as part of the process of adding other items to the database. There is also a new button on the main form for generating interviews. The .sql file that this button uses will be discussed later on in the paper.

The image shows a Windows-style dialog box titled "Add Job Fair". It has a standard title bar with minimize, maximize, and close buttons. The main area contains several labels and input controls:
 

- Title**: A text input field.
- Description**: A text input field.
- Start Date**: A text input field containing the placeholder "YYYY-MM-DD".
- End Date**: A text input field containing the placeholder "YYYY-MM-DD".
- Website**: A text input field.
- Phone**: A text input field.
- Contact Person**: A dropdown menu with a downward arrow.
- Interview Start Time**: A dropdown menu with a downward arrow.
- Interview End Time**: A dropdown menu with a downward arrow.

 At the bottom right of the dialog is a button labeled "Add Job Fair".

I've updated my form for adding job fairs since the last progress report. I started by automating data entry into the JobFairDays table. I use the start and end date provided by the user, and use a for loop to add each day to the table after the job fair itself is added.

```
// This script borrowed from https://www.dbrnd.com/2015/08/list-all-dates-between-two-dates-in-sql-server/
```

```
string command = "DECLARE @StartDate DATE " +
  "DECLARE @EndDate DATE " +
  "SET @StartDate = '" + tbStartDate.Text + "' " +
  "SET @EndDate = '" + tbEndDate.Text + "'; " +
  "WITH DateRange(DateData) AS ( " +
  "SELECT @StartDate as Date " +
  "UNION ALL " +
  "SELECT DATEADD(d, 1, DateData) " +
  "FROM DateRange " +
  "WHERE DateData < @EndDate) " +
  "SELECT DateData " +
  "FROM DateRange;";

List<string> dates = Table.Read(command, 1);
jobFairDays.Columns[1].CurrentValue = jobFairs.Columns[0].CurrentValue;
timeSlots.Columns[1].CurrentValue = jobFairs.Columns[0].CurrentValue;

// Variables for entering time slot data
int minute = 0;
string startTime;
string endTime;
startHour = offset + cbIntStart.SelectedIndex;
```

```
endHour = offset + 1 + cbIntStart.SelectedIndex + cbIntEnd.SelectedIndex;
```

```
for (int i = 0; i < dates.Count; i++)
{
    jobFairDays.Columns[2].CurrentValue = dates[i];
    jobFairDays.InsertRow();
}
```

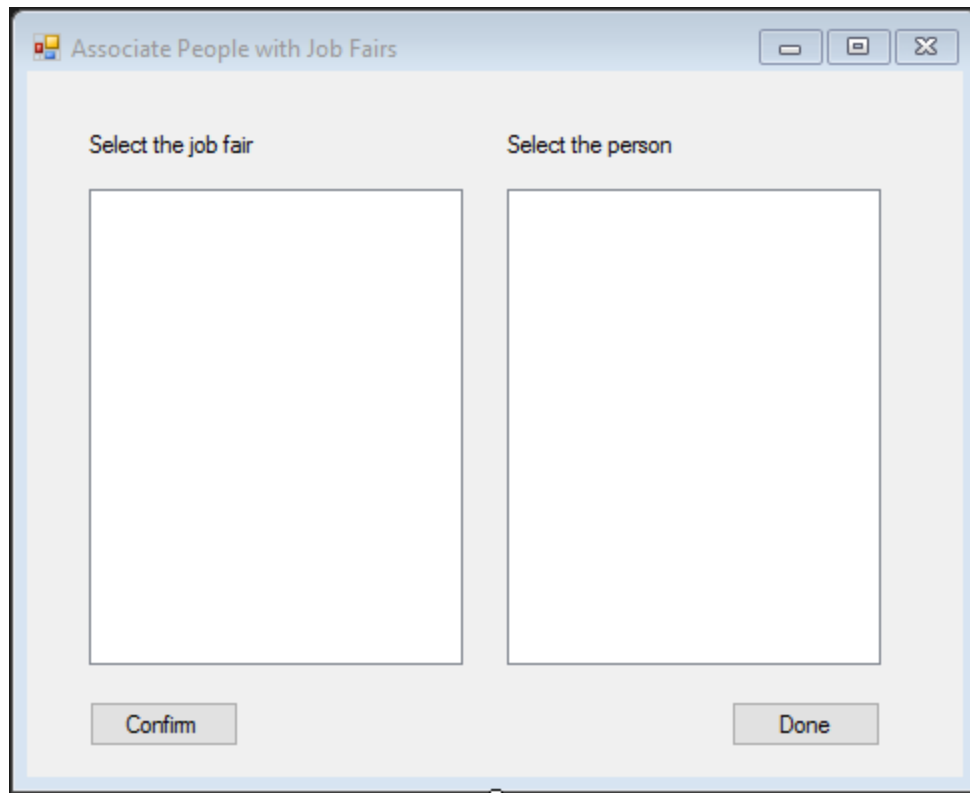
The rest of this for loop is dedicated to adding time slots for each job fair day.

I've also added combo boxes for selecting interview start and end times so the user can decide what time range interviews can take place in. The default is 9:00 AM to 5:00 PM, but any positive range can be selected. When the user changes the interview start time, the end time combo box is updated to only include times after the start time. Following is my for loop for adding time slots to the database using input from the job fair form.

```
for (int i = 0; i < dates.Count; i++)
{
    jobFairDays.Columns[2].CurrentValue = dates[i];
    jobFairDays.InsertRow();

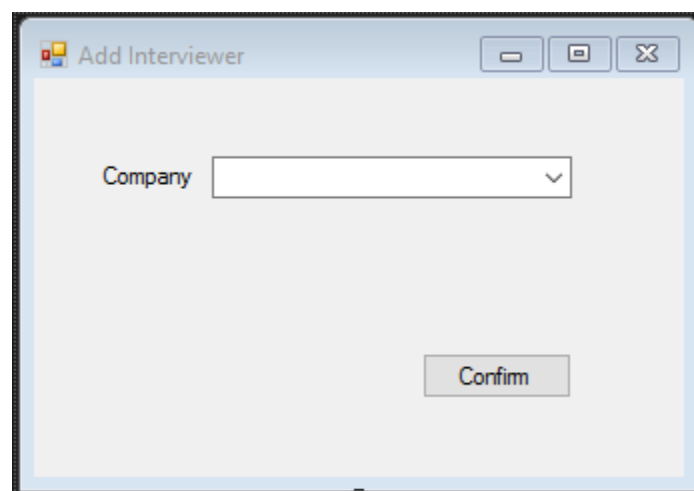
    for (int hour = startHour; hour < endHour; hour++)
    {
        timeSlots.Columns[2].CurrentValue = jobFairDays.Columns[0].CurrentValue;
        endTime = hour + ":" + minute.ToString("00") + ":00";

        while (minute <= (60 - interval)) {
            startTime = endTime;
            minute += interval;
            if (minute == 60)
                endTime = (hour + 1) + ":00:00";
            else
                endTime = hour + ":" + minute.ToString("00") + ":00";
            timeSlots.Columns[3].CurrentValue = startTime;
            timeSlots.Columns[4].CurrentValue = endTime;
            timeSlots.InsertRow();
        }
        minute = minute % 60;
    }
}
```



The dialog box titled "Associate People with Job Fairs" features a light blue header bar with standard window controls (minimize, maximize, close). The main area is divided into two columns. The left column is headed "Select the job fair" and contains a large, empty rectangular box. The right column is headed "Select the person" and also contains a large, empty rectangular box. At the bottom of the dialog, there are two buttons: "Confirm" on the left and "Done" on the right.

After I add a person, I show this form to associate a person with job fairs. This form can also be shown by clicking the appropriate button on the main form. This form is very simple and follows the same patterns as most of my data entry forms. If a person ID is passed into the constructor, the person label and list box are hidden. Otherwise, both are shown, and the user can associate any person with any job fair.

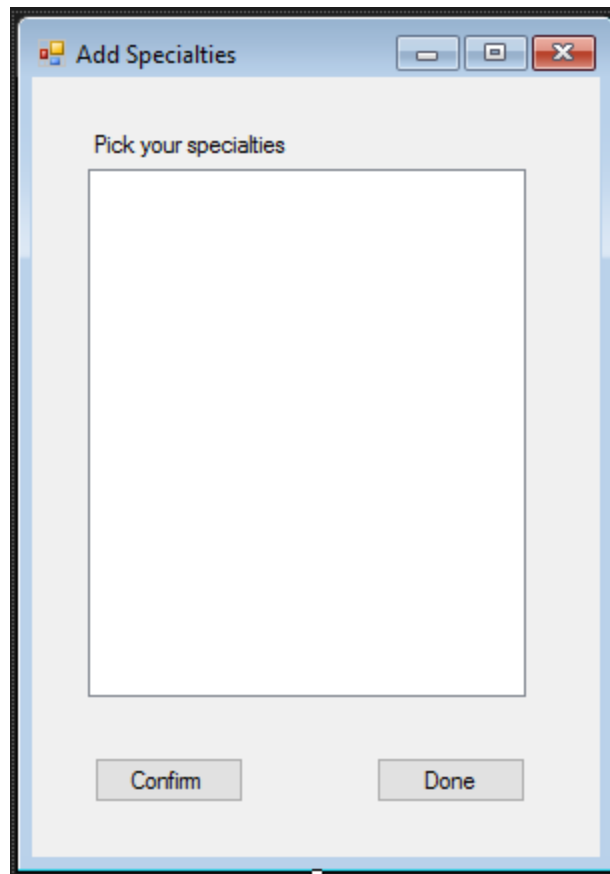


The dialog box titled "Add Interviewer" has a light blue header bar with standard window controls. The main area contains a label "Company" followed by a text input field with a dropdown arrow on the right. At the bottom center of the dialog is a single "Confirm" button.

Interviewers, candidates, and locations are added to the database in the same way that most items are added. The only difference is that my `formAddInterviewer` constructor

contains a parameter called `personID` that I use as the `PersonID` column's current value. The `formAddCandidate` and `formAddLocation` constructors are similar.

```
public formAddInterviewer(int personID)
{
    InitializeComponent();
    StartPosition = FormStartPosition.CenterParent;
    SetUpTable();
    interviewers.Columns[1].CurrentValue = personID.ToString();
}
```



After I added my interviewer, candidate, and location forms, I created a form for adding specialties. There is just one form for candidates and interviewers. In my constructor, I pass in a candidate ID and an interviewer ID, using -1 as a flag to indicate that there is no candidate or no interviewer.

```
public formAddPersonSpecialties(int candidateID, int interviewerID)
{
    InitializeComponent();
    StartPosition = FormStartPosition.CenterParent;
```

```

SetUpTable();
candidateSpecialties.Columns[0].CurrentValue = candidateID.ToString();
interviewerSpecialties.Columns[0].CurrentValue = interviewerID.ToString();

if (candidateID != -1)
    labelSpecialties.Text = "Select your candidate specialties.";
else
    labelSpecialties.Text = "Select your interviewer specialties.";
}

```

Then, when the Confirm button is clicked, I check the current value of the candidateSpecialties table to see if specialties are being added to a candidate or to an interviewer. I use that knowledge to insert data into the correct table.

```

private void btnConfirm_Click(object sender, EventArgs e)
{
    if (candidateSpecialties.Columns[0].CurrentValue != "-1")
    {
        candidateSpecialties.Columns[1].CurrentValue =
            lbSpecialties.SelectedValue.ToString();
        candidateSpecialties.InsertRow();
    }
    else
    {
        interviewerSpecialties.Columns[1].CurrentValue =
            lbSpecialties.SelectedValue.ToString();
        interviewerSpecialties.InsertRow();
    }
}

```

The image shows a Windows-style dialog box titled "Add Tables". It has a standard title bar with minimize, maximize, and close buttons. The main area contains two dropdown menus, one labeled "Job fair" and one labeled "Location". Below these are two checkboxes, "Accessible" and "Has power", both of which are currently unchecked. At the bottom of the dialog, there are two buttons: "Add Table" on the left and "Done" on the right.

In my `formAddTables` constructor, I pass in a job fair ID and hide the Job Fair combo box if the ID is valid. I pass in -1 as an invalid ID if there is no current job fair. I fill the Location combo box by reading location data from the database to get all of the locations associated with the current job fair. If there is no valid job fair ID passed in, I update the Location combo box every time the Job Fair combo box's selected index is changed. I do this as follows:

```
string command = "SELECT l.Name, l.ID " +
    "FROM Locations AS l "
    + "WHERE JobFairID = " + jobFairID + ";";
List<string> locations = Table.Read(command, 2);

cbLocation.Items.Clear();

for (int i = 0; i < locations.Count; i += 2)
{
    Column thisColumn = new Column(locations[i]);
    thisColumn.CurrentValue = locations[i + 1];
    cbLocation.Items.Add(thisColumn);
}
```



## Interview Automation Algorithm

### *Overview*

My algorithm generates a table called Pairs that matches candidates to interviewers. Specialties and desired employment types are considered in this pairing. I create a table called Environments that crosses time slots with available tables. Then, I iterate through the Pairs table and match each Pair with an Environment where the interview will take place. These results are saved in the Interviews and InterviewInterviewers tables in the JobFair database.

### *Code*

The first step to generating the Pairs table is generating an intermediate table called TheseCandidates. It includes a JobFairID, PersonID, SpecialtyID, and EmploymentTypeID. This generates as many rows per candidate as there are specialties for that candidate.

```
/* Get all the candidates and their specialties */

DROP TABLE IF EXISTS #TheseCandidates

CREATE TABLE #TheseCandidates
(
    JobFairID int,
    PersonID int,
    SpecialtyID int,
    EmploymentTypeID int
);

INSERT INTO #TheseCandidates

SELECT jfp.JobFairID,
       jfp.PersonID,
       cs.SpecialtyID,
       c.EmploymentTypeID
FROM [JobFair].[dbo].[JobFairPeople] AS jfp
INNER JOIN [JobFair].[dbo].[Candidates] AS c
ON jfp.PersonID = c.PersonID
INNER JOIN [JobFair].[dbo].[CandidateSpecialties] AS cs
ON c.ID = cs.CandidateID;
;
```

I use similar code to create a table called TheseInterviewers. The only difference is that the TheseInterviewers table includes a CompanyID.

```
/* Get all the interviewers and their specialties */
```

```

DROP TABLE IF EXISTS #TheseInterviewers

CREATE TABLE #TheseInterviewers
(
    JobFairID int,
    PersonID int,
    SpecialtyID int,
    CompanyID int,
    EmploymentTypeID int
);

INSERT INTO #TheseInterviewers

SELECT jfp.JobFairID, jfp.PersonID, iSpec.SpecialtyID, i.CompanyID, cet.EmploymentTypeID
FROM [JobFair].[dbo].[JobFairPeople] AS jfp
INNER JOIN [JobFair].[dbo].[Interviewers] AS i
ON jfp.PersonID = i.PersonID
INNER JOIN [JobFair].[dbo].[InterviewerSpecialties] AS iSpec
ON i.ID = iSpec.InterviewerID
INNER JOIN [JobFair].[dbo].[CompanyEmploymentTypes] AS cet
ON cet.CompanyID = i.CompanyID;

```

Next, I use these two tables to create the Pairs table. Wherever interviewers and candidates have the same specialties and desired employment types for a specific job fair, a Pair is created.

```

/* Create pairings of candidates/interviewers where their specialties overlap */

DROP TABLE IF EXISTS #Pairs

CREATE TABLE #Pairs
(
    InterviewerPersonID int,
    CandidatePersonID int,
    CompanyID int,
    SpecialtyID int,
    EmploymentTypeID int
);

INSERT INTO #Pairs

SELECT ti.PersonID AS InterviewerPersonID,
tc.PersonID AS CandidatePersonID,
ti.CompanyID,
ti.SpecialtyID,
ti.EmploymentTypeID
FROM #TheseInterviewers AS ti
INNER JOIN #TheseCandidates AS tc
ON (ti.SpecialtyID = tc.SpecialtyID AND ti.EmploymentTypeID = tc.EmploymentTypeID)
ORDER BY CompanyID;

```

Then, I move on to creating the Environments table. This is a list of time slots matched with each table that the time slot is valid for. The Environments table has all of the information about where and when the interview is.

```

/* Generate time slot/table pairs */

DROP TABLE IF EXISTS #Environments

CREATE TABLE #Environments
(
    TimeSlotID int,
    JobFairID int,
    DayID int,
    JobFairDate date,
    StartTime time(0),
    EndTime time(0),
    LocationID int,
    TableID int
)

INSERT INTO #Environments

SELECT ts.ID,
       ts.JobFairID,
       ts.DayID,
       jfd.Date,
       ts.StartTime,
       ts.EndTime,
       t.LocationID,
       t.ID
FROM [JobFair].[dbo].[TimeSlots] AS ts
INNER JOIN [JobFair].[dbo].[JobFairDays] AS jfd
ON ts.DayID = jfd.ID AND ts.JobFairID = jfd.JobFairID
INNER JOIN [JobFair].[dbo].[Tables] AS t
ON t.JobFairID = ts.JobFairID;

```

Finally, I match Pairs with Environments to create Interviews. I start by creating a table called PersonTimes. This is for keeping track of which people are occupied at which times.

```

DROP TABLE IF EXISTS #PersonTimes

CREATE TABLE #PersonTimes (
    PersonID int,
    StartTime time
)

```

Then, I declare all the variables I need for the procedure. These are the variables that hold data from the Pairs and Environments tables.

```

/* #Pairs data */
DECLARE @InterviewerPersonID int;
DECLARE @CandidatePersonID int;
DECLARE @CompanyID int;
DECLARE @SpecialtyID int;
DECLARE @EmploymentTypeID int;

/* #Environments data */
DECLARE @TimeSlotID int;
DECLARE @JobFairID int;
DECLARE @JobFairDate date;
DECLARE @StartTime time(0);
DECLARE @EndTime time(0);
DECLARE @LocationID int;
DECLARE @TableID int;
DECLARE @InterviewID int;

```

I iterate through each row in Pairs, select the first row, and save its data in the appropriate variables.

```

WHILE EXISTS (SELECT DISTINCT * FROM #Pairs)
BEGIN

    SELECT TOP 1
        @InterviewerPersonID = InterviewerPersonID,
        @CandidatePersonID = CandidatePersonID,
        @CompanyID = CompanyID,
        @SpecialtyID = SpecialtyID,
        @EmploymentTypeID = EmploymentTypeID
    FROM #Pairs
    ORDER BY CompanyID, InterviewerPersonID ASC

```

Then, I iterate through each row in Environments. I again select the top row and save its data in the appropriate variables.

```

WHILE EXISTS (SELECT * FROM #Environments)
BEGIN

    SELECT TOP 1
        @TimeSlotID = #Environments.TimeSlotID,
        @JobFairID = #Environments.JobFairID,
        @JobFairDate = #Environments.JobFairDate,
        @StartTime = #Environments.StartTime,
        @EndTime = #Environments.EndTime,
        @LocationID = #Environments.LocationID,
        @TableID = #Environments.TableID
    FROM #Environments
    ORDER BY JobFairDate, StartTime ASC;

```

If the candidate and interviewer are not busy at the selected time, I assign that Environment to the current Pair. I fill in the PersonTimes table with the relevant data,

create a row in the Interviews table, and use the returned identity value to create a new row in the InterviewInterviewers table.

```

IF NOT EXISTS(SELECT * FROM #PersonTimes
WHERE (#PersonTimes.PersonID = @InterviewerPersonID
OR #PersonTimes.PersonID = @CandidatePersonID)
AND #PersonTimes.StartTime = @StartTime)
BEGIN

    INSERT INTO #PersonTimes
        (PersonID, StartTime)
    VALUES (@InterviewerPersonID, @StartTime);
    INSERT INTO #PersonTimes
        (PersonID, StartTime)
    VALUES (@CandidatePersonID, @StartTime);
    INSERT INTO [JobFair].[dbo].[Interviews]
        (JobFairID, CandidateID, TimeSlotID, TableID,
        CompanyID, EmploymentTypeID, SpecialtyID)
    VALUES (@JobFairID, @CandidatePersonID, @TimeSlotID,
            @TableID, @CompanyID, @EmploymentTypeID,
            @SpecialtyID);

    SET @InterviewID = @@IDENTITY;

    INSERT INTO [JobFair].[dbo].[InterviewInterviewers]
        (InterviewID, InterviewerID)
    VALUES (@InterviewID, @InterviewerPersonID);

    DELETE FROM #Environments
    WHERE #Environments.TimeSlotID = @TimeSlotID
        AND #Environments.LocationID = @LocationID
        AND #Environments.TableID = @TableID

```

Finally, I delete the used rows from Environments and Pairs and break out of the inner while loop. If either the candidate or interviewer is busy at the selected time, the loop moves to the next time slot until one is found.

```

DELETE FROM #Pairs
WHERE #Pairs.InterviewerPersonID = @InterviewerPersonID
    AND #Pairs.CandidatePersonID = @CandidatePersonID
    AND #Pairs.CompanyID = @CompanyID
    AND #Pairs.SpecialtyID = @SpecialtyID
    AND #Pairs.EmploymentTypeID = @EmploymentTypeID

BREAK;

END

END

END

```

## Results

When I run the algorithm from the job fairs app, I expect one interview to be generated, and it is generated accurately, as shown on the next page. The InterviewInterviewers table is also populated with one row showing the interviewer for this interview.

	ID	JobFairID	CandidateID	TimeSlotID	TableID	CompanyID	EmploymentTypeID	SpecialtyID
1	10	2030	2006	1	2	1	3	1

### Next Steps

Although I have the bulk of the work done in allowing for data entry and generating interviews, I can still see a lot of work that could be done to improve my app. Following is a general list of the steps I plan to take moving forward.

- Sanitize database inputs in the newer forms. It is currently possible to get error messages by trying to insert data with duplicate primary keys from newer forms. I will have to read the rows already in the database and check the user's data against those rows.
- Enhance my interview generation algorithm. Interviews should allow for multiple interviewers at the same interview, and right now I do not allow that functionality. To do this, I will check to see if the next Pair in the table has the same CandidateID and CompanyID as the current row, and if so, include that interviewer and specialty in the current interview.
- Allow for data editing and deletion from the app. Right now, data can be added, but editing and deleting data must be done through SSMS.
- Clean up my code. The code currently does not have a lot of comments. I should add comments for clarity and make sure no code is redundant or poorly written.

### Conclusion

After I modified my code as discussed in Progress Report 1 to include a Table and Column class, the rest of the data entry forms were easy to create. The interesting part of the work done for this progress report was writing the interview generation algorithm. The first part (creating the Pairs and Environments tables) was relatively easy. I had gotten some practice with select statements from code I had already written for this app, so the code for that part was familiar to me. However, I haven't worked with .sql scripts before, so I was exploring new parts of a new language while writing the part of my algorithm that matched Pairs to Environments. I had to adjust to the decidedly non-C-based syntax, which did confuse me a lot at first. I found some helpful guides on how to iterate through rows in a table, and by adapting those to my purpose, I was able to finish the algorithm relatively painlessly. The most confusing error I got while working on my interview generation algorithm was one that said that one of my temporary tables had a different number of columns than the number of columns I was attempting to insert. This happened whenever I tried to execute the script after altering the structure of the table. I found that I had to drop the table before running the script again, even though the script

contained a drop table command, in order to solve this error. My script seems to be working right now, but it feels hacked together, and I would guess that there are best practices that I'm not adhering to. I would be interested to know how I could improve the script to be more efficient and to adhere to best practices.