Creating the Programming Environments Application
Programming Environments
Andreya Grzegorzewski – Ohio Northern University
Fall 2017-2018

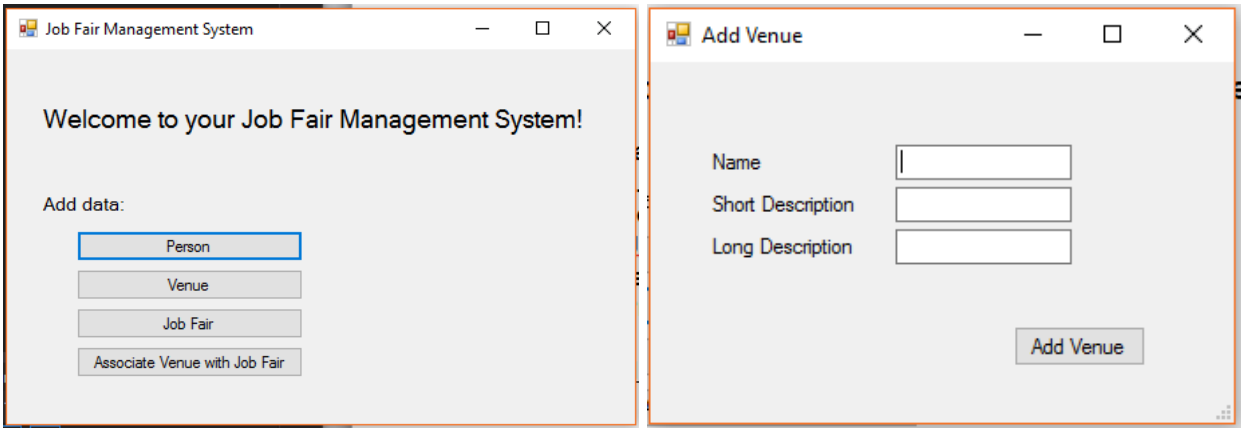
Introduction

By the end of the Programming Environments course, each student should have an application that will allow the user to do a variety of tasks related to job fair management. The first task this application is meant to cover is data entry. At this point in the semester, the application should allow the user to enter data into the Venues, People, and JobFairs tables. My app also allows for data entry into the JobFairVenues table, which is necessary to link tuples in the Venues table to tuples in the JobFairs table. This assignment documents my work on my Programming Environments app so far. The complete code for this application is available on my GitHub account in the programming-environments repository.

Narrative and Code

Independent Data Entry

I started creating my app by designing the forms I would need. The main form, whose title text is “Job Fair Management System,” allows the user to select which data he wants to add. Next, I created the forms for adding people, venues, and job fairs. Finally, I created the form to associate a venue with a job fair. Each data entry form is composed of a series of labels, each of which is associated with a text box or a combo box. The main form, Form1, and the venue data entry form, formAddVenue, are shown below.



Once I had my forms created, I worked on the code for Form1. I first connected my app to the JobFair database, made a connection string variable, and created a getter for that variable.

```
static string connStr = @"Data Source=localhost\SQLEXPRESS;Initial Catalog = JobFair;
    Integrated Security = True";

public static string getConnStr()
{
    return connStr;
}
```

Then, I handled the button click events for Form1. For each button click event generated in Form1, I created and showed the appropriate form.

```
private void buttonAddVenue_Click(object sender, EventArgs e)
{
    formAddVenue form = new formAddVenue();
    form.ShowDialog(this);
}
```

In order to enter a venue into the Venues table, I needed to create a Venue class. I gave it one member variable for each column in the Venues table.

```
class Venue
    {
        private int id;
        private string name = "";
        private string shortDesc = "";
        private string longDesc = "";
```

I created getters and setters for each field. An example is shown below.

```
public int ID
{
    get { return id; }
    set { id = value; }
}
```

Finally, I created an Insert method.

```
public bool Insert()
{
    // Connect to the database
    string connStr = Form1.getConnStr();
    SqlConnection sc = new SqlConnection(connStr);
    sc.Open();

    // Create Insert command
```

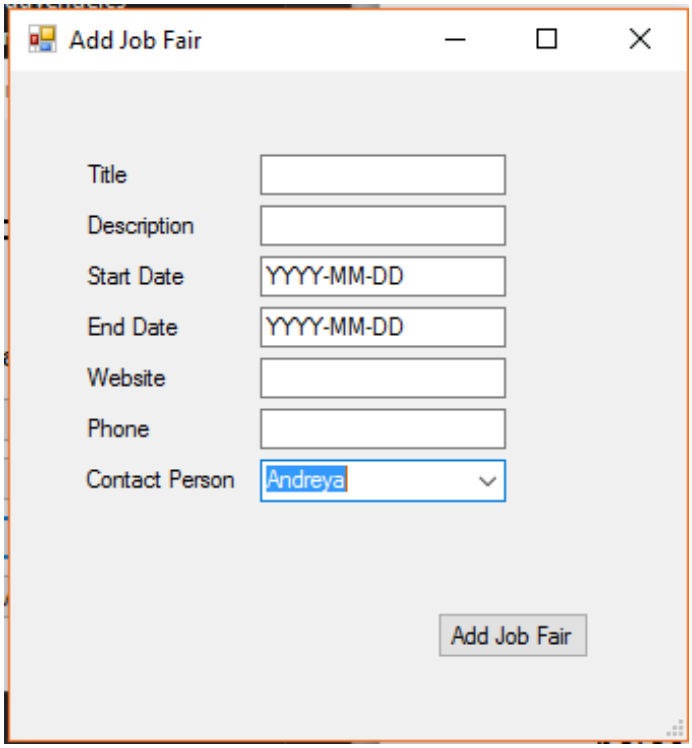
```
    SqlCommand cmd = new SqlCommand();
    cmd.CommandText = "INSERT INTO Venues " +
        "(Name, [Short Description], [Long Description]) VALUES ('" +
        name + "', '" +
        shortDesc + "', '" +
        longDesc + "')";

    // Set command data
    cmd.CommandType = System.Data.CommandType.Text;
    cmd.Connection = sc;

    // If the command works, return true
    if (cmd.ExecuteNonQuery() == 1)
    {
        sc.Close();
        return true;
    }

    // Otherwise, return false
    sc.Close();
    return false;
}
```

I repeated this process for the People and JobFairs tables. The code for these forms and classes is available in the JobFairs folder of my programming-environments repository on GitHub, which was linked to in the Introduction section.

At this point, I had an “Add Venue” form and a Venue class for inserting the data into the database, but I hadn’t linked these two elements together. I finished by adding an event handler for the “Add Venue” button, then repeated the process by adding event handlers to the “Add”.

```
private void buttonAddVenue_Click(object sender, EventArgs e)
{
    // Create the Venue object
    Venue v = new Venue();
    v.Name = tbName.Text;
    v.ShortDescription = tbShortDesc.Text;
    v.LongDescription = tbLongDesc.Text;

    // Insert it and let the user know if the insertion worked
    if (v.Insert())
    {
        MessageBox.Show(this, v.Name + " was added.", "Success!");
        this.Close();
    }
    else
    {
        MessageBox.Show(this, v.Name + " could not be added.", "Error");
    }
}
```

Data Sources for Combo Boxes



Some tables have foreign keys, or fields that identify a tuple from another table. This occurs in the JobFairs table; the ContactPersonID field is a foreign key for the ID field in the Person table. In order to facilitate the selection of the correct value from a list of foreign keys, I used combo boxes. The display member is a string that has meaning to a person; in this form, the display member is the person's first name. The value member is the person's ID.

To populate the combo box, I created a DataSource for it in the Properties tab. The DataSource for the Contact Person combo box is called peopleBindingSource. Because the data source populates the combo box in the background, I didn't have to write any code to do this.

In order to get the selected ID from the combo box, I had to create a DataRow object from the selected item in the combo box, then extract the ID field from the DataRow.

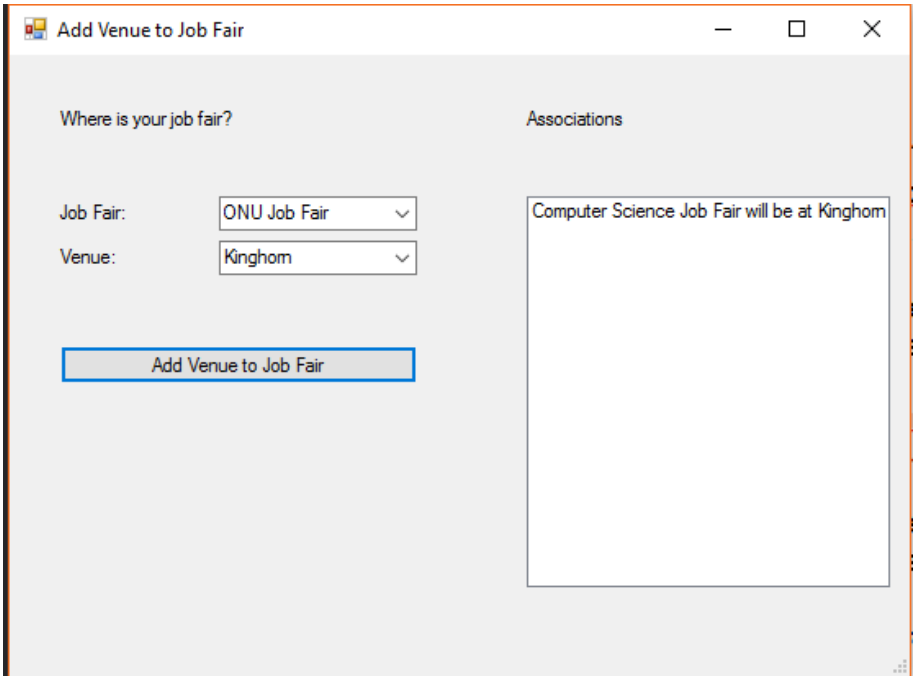
```
DataRow contact = ((DataRowView)cbContact.SelectedItem).Row;
j.ContactPersonID = int.Parse(contact["ID"].ToString());
```

The rest of the code for adding a job fair to the database is consistent with the code for adding a person or a venue.

Associating Venues with Job Fairs

Job fairs could potentially take place at more than one venue, and one venue could host multiple job fairs. Therefore, the JobFair database has a table called JobFairVenues that allows for multiple venue-job fair associations. To facilitate data entry into this table, I created a form that allows the user to create associations between job fairs and venues. The form, shown on the next page, appears after a job fair is added to the database. The user can also access this form from Form1, so he can create venue-job fair associations even if the job fair has already been entered into the database. This

form also has a list box that shows already-existing associations between job fairs and venues.



The image to the left is what the form looks like when it is accessed from Form1. However, if it appears after a job fair is entered into the database, the Job Fair label and combo box are hidden, and only associations relevant to the current job fair can be created. Additionally, only associations for the current job fair are shown in the list box. The combo boxes in this form have data sources associated with them as discussed earlier in the *Data Sources for Combo Boxes* section.

After I designed the form, I created some member variables.

```
// Start by assuming we aren't working with a specific job fair
// And no venue is selected
private int jobFairID = -1;
private string jobFairTitle = "";
private int venueID = -1;
private bool specificJobFair = false;
```

Then, I created a method that allows the current job fair to be specified. If the form is generated after a job fair is entered into the database, this method is called and the form is updated to reflect the present conditions.

```
// Set the current job fair to the one just entered into the DB
public void SpecifyJobFair(JobFair j)
{
    // Only allow associations for the current job fair
    cbJobFair.Visible = false;
    labelJobFair.Visible = false;
    labelTitle.Text = "Where is " + j.Title.Replace("''", "'") + "?";

    // Update variables to reflect the current job fair
    jobFairTitle = j.Title.Replace("''", "'");
```

```
    jobFairID = j.ID;
    specificJobFair = true;

    // Remove any associations that aren't for this job fair
    for (int i = 0; i < lbAssociations.Items.Count; i++)
    {
        if (!lbAssociations.Items[i].ToString().Contains(jobFairTitle))
        {
            lbAssociations.Items.RemoveAt(i);
            i--;
        }
    }
}
```

Next, I created a method to fill the list box with venue-job fair associations.

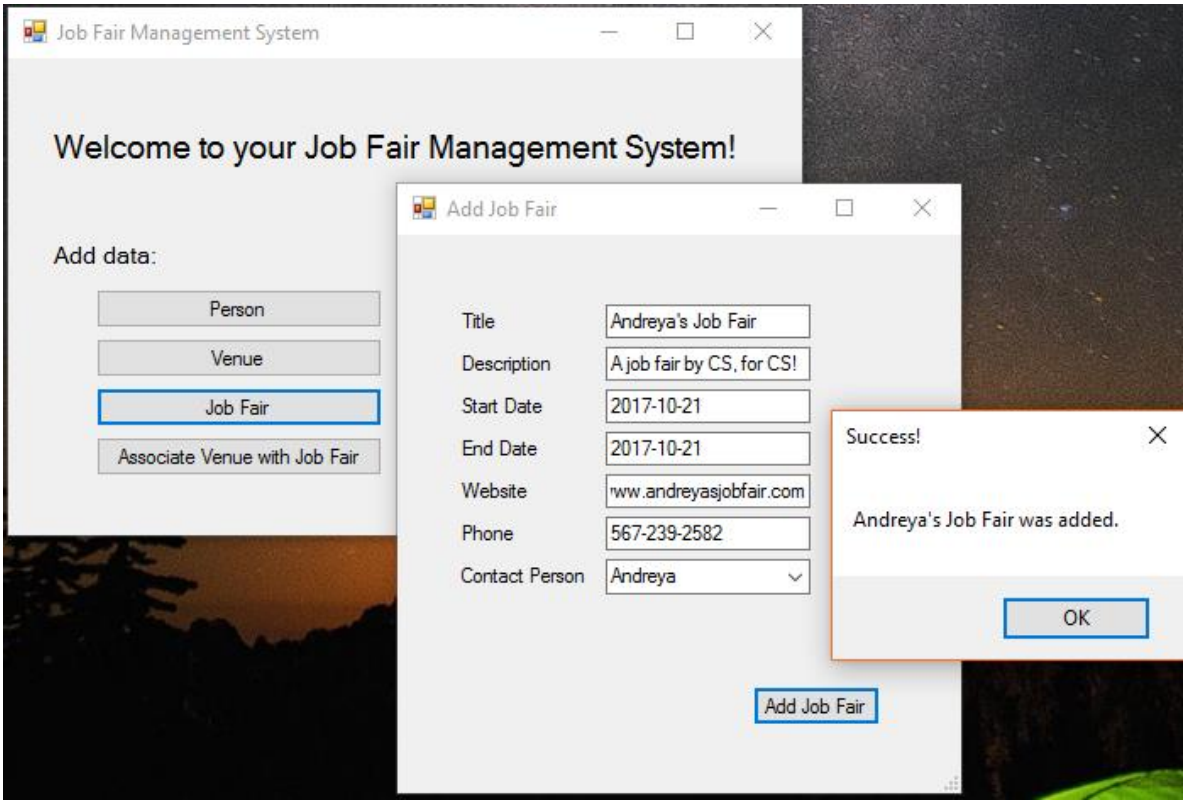
```
// Show the user all venue-job fair associations that already exist
private void FillListBox()
{
    // Connect to the DB
    string connStr = Form1.getConnStr();
    SqlConnection sc = new SqlConnection(connStr);
    sc.Open();

    // Select job fair title/venue name pairs
    SqlCommand cmd = new SqlCommand();
    cmd.CommandText = "SELECT j.Title, v.Name " +
        "FROM JobFairVenues AS jv " +
        "INNER JOIN JobFairs AS j " +
        "ON j.ID = jv.JobFairID " +
        "INNER JOIN Venues AS v " +
        "ON v.ID = jv.VenueID;";

    // Set command data
    cmd.CommandType = System.Data.CommandType.Text;
    cmd.Connection = sc;

    // The current job fair title and venue name
    string thisTitle;
    string thisName;

    // Iterate through title/name pairs
    using (SqlDataReader reader = cmd.ExecuteReader())
    {
        while (reader.Read())
        {
            thisTitle = reader.GetString(0);
            thisName = reader.GetString(1);

            // If the pair should be shown, show it
            if (!specificJobFair || thisTitle == jobFairTitle)
                lbAssociations.Items.Add(thisTitle + " will be at " + thisName);
        }
    }
    sc.Close();
}
```

Finally, I wrote the code for the click event handler for the “Add Venue to Job Fair” button.

```
private void buttonAdd_Click(object sender, EventArgs e)
{
    // Get the select job fair and venue
    DataRow jobFair = ((DataRowView)cbJobFair.SelectedItem).Row;
    DataRow venue = ((DataRowView)cbVenue.SelectedItem).Row;

    // If we don't know which job fair we're working with, get the ID
    if (!specificJobFair)
    {
        jobFairID = int.Parse(jobFair["ID"].ToString());
    }

    // Get the venue ID
    venueID = int.Parse(venue["ID"].ToString());

    // Connect to the DB
    string connStr = Form1.getConnStr();
    SqlConnection sc = new SqlConnection(connStr);
    sc.Open();

    // Create insert statement and set command data
    SqlCommand cmd = new SqlCommand();
    cmd.CommandText = "INSERT INTO JobFairVenues " +
        "(JobFairID, VenueID) VALUES ('" +
        jobFairID + "', '" +
        venueID + "')";
    cmd.CommandType = System.Data.CommandType.Text;
    cmd.Connection = sc;

    // Try to add the association
    if (cmd.ExecuteNonQuery() != 1)
        MessageBox.Show(this, "The venue could not be added to the job fair.", "Error");

    // Close the DB connection and update the list box
    sc.Close();
    lbAssociations.Items.Clear();
    FillListBox();
}
```

The Deliverable

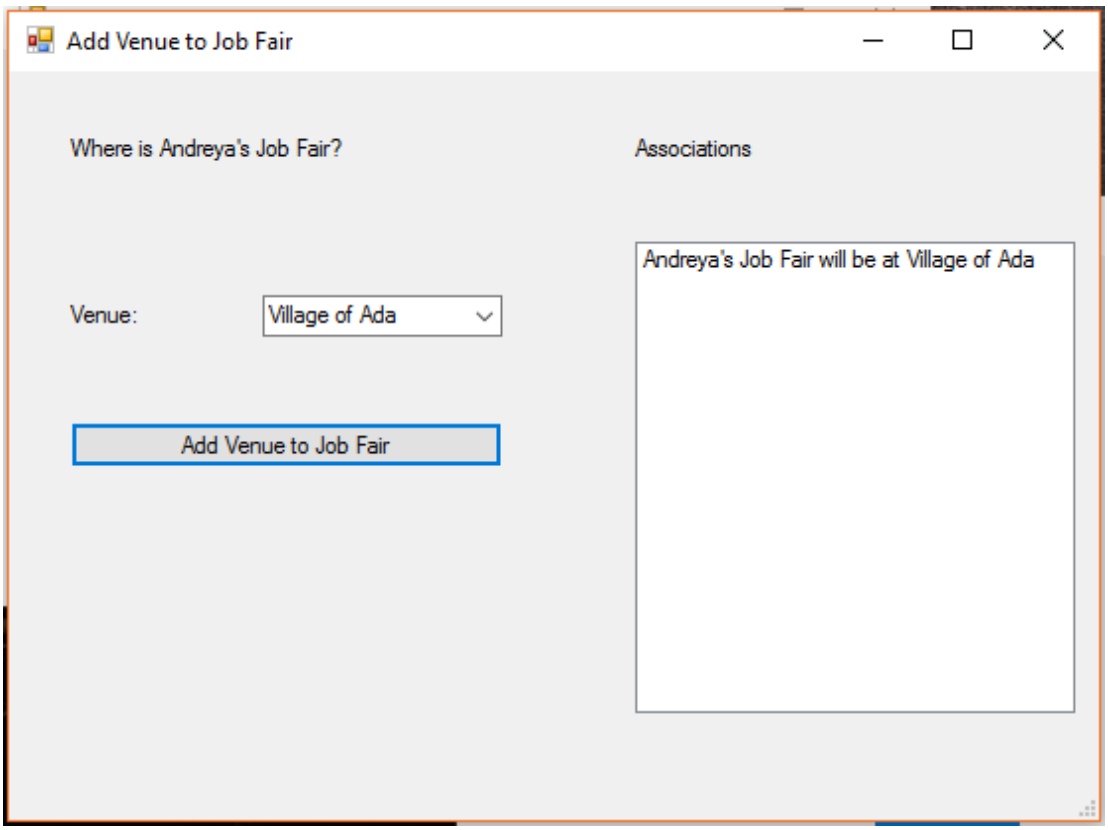
On the next page is a screenshot of my app after I add “Andreya’s Job Fair” to the database. After clicking the “Add Job Fair” button, a success message appears.



The new job fair was added to the JobFairs table in the JobFair database. All rows in the JobFairs table are shown below, with the newly added job fair highlighted.

	ID	Title	Description	StartDate	EndDate	Website	Phone	ContactPersonID
1	1	ONU Job Fair		2017-09-21	2017-09-22			2
2	2	Computer Science Job Fair	Get that CS job!	2017-10-10	2017-10-10			3
3	3	Honda Info Session	Info Session for Honda	2017-12-13	2017-12-14			3
4	22	Andreya's Job Fair	A job fair by CS, for CS!	2017-10-21	2017-10-21	www.andreyasjobfair.com	567-239-2582	2

When I clicked the “OK” button on the success message, the form to associate job fairs with venues appears. I associated Andreya’s Job Fair with the Village of Ada. The form, after the association was added, is shown on the next page.




```
SELECT j.Title, v.Name
    FROM JobFairVenues AS jv
    INNER JOIN JobFairs AS j
    ON j.ID = jv.JobFairID
    INNER JOIN Venues AS v
    ON v.ID = jv.VenueID;
```

100 %

Results | Messages

	Title	Name
1	Computer Science Job Fair	Kinghorn
2	Andreya's Job Fair	Village of Ada

To the left is the query to select job fair title and venue name pairs. The second item in the list is the association just added by the “Add Venue to Job Fair” form.

Observations

This app must facilitate a lot of repetitive actions. For example, the process of adding a person to the database is very similar to the process of adding a venue to the database. I followed process to add data to the database that was discussed in class. In that process, an object is created, and an Insert method is called on that object. However, this process could probably be generalized. Instead of creating classes for each table that constitutes a “thing,” I could create a “Table” class to represent a table in the

database, and a “Field” class to represent a field in a table. This could potentially simplify the code a lot and reduce some of the redundancy.

Although I didn’t show the code for it in this paper, I did some input sanitation before inserting data into the database. For example, I created a method to ensure that date entries are formatted properly, and the app ensures that a person’s middle initial is only one character. However, there is a lot more sanitation that I could do. Some tricky data enterer could type a command to delete a table into a text box, and the table would be deleted. This is something that would need to be considered if this was a published and monetized app.