```
/*-----------------------------------------------------------------------------------------------
 * Author:      Dan Cassidy
 * Date:        2015-06-23
 * Assignment:  cView-P4
 * Source File: Default.aspx.cs
 * Language:    C#
 * Course:      CSCI-C 490, C# Programming, MoWe 08:00
 * Project:     The overall goal of this project is to capitalize on the fact that government, from
 *              local to national, has made some of its data open by developing a way to explore
 *              this data and present it to a user in a meaningful fashion. This phase of the
 *              project is meant to explore data from any combination of the Business dataset
 *              (https://data.southbendin.gov/d/imxu-7m5i), the Parks and Features dataset
 *              (https://data.southbendin.gov/d/yf5x-7tkb), and the Public Facility dataset
 *              (https://data.southbendin.gov/d/jeef-dsq9) using a web-based ASP.NET UI interacting
 *              with a SQL backend via the Entity Framework.
 * Purpose:     Code-behind file for Default.aspx.  This file's only purpose is to redirect the
 *              client to the Main.aspx page.
-----------------------------------------------------------------------------------------------*/

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace cView_P4_DanCassidy
{
    public partial class Default : System.Web.UI.Page
    {
        /*-----------------------------------------------------------------------------------------
         * Name:    Page_Load
         * Type:    Event Handler Method
         * Purpose: Handles anything that should happen on page load.  In this case, it shunts the
         *          client over to the Main.aspx page instead of this one.
         * Input:   object sender, holds a reference to the object that raised this event.
         * Input:   EventArgs e, holds data related to this event.
         * Output:  Nothing.
        -----------------------------------------------------------------------------------------*/
        protected void Page_Load(object sender, EventArgs e)
        {
            Server.Transfer("~/Main.aspx");
        }
    }
}
```

```
/*-----------------------------------------------------------------------------------------------
 * Author:      Dan Cassidy
 * Date:        2015-06-23
 * Assignment:  cView-P4
 * Source File: Main.aspx.cs
 * Language:    C#
 * Course:      CSCI-C 490, C# Programming, MoWe 08:00
 * Purpose:     Code-behind file for Main.aspx. Controls randomization and resetting of the tables
 *              behind the application.
-----------------------------------------------------------------------------------------------*/

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace cView_P4_DanCassidy
{
    public partial class Menu : System.Web.UI.Page
    {
        /*-----------------------------------------------------------------------------------------
         * Name:    btnRandomize_Click
         * Type:    Event Handler Method
         * Purpose: Handles randomizing the tables when clicked.
         * Input:   object sender, holds a reference to the object that raised this event.
         * Input:   EventArgs e, holds data related to this event.
         * Output:  Nothing.
        -----------------------------------------------------------------------------------------*/
        protected void btnRandomize_Click(object sender, EventArgs e)
        {
            // Hide things until needed.
            lblResult.Visible = false;
            lblError.Visible = false;

            try
            {
                using (CViewDataEntities db = new CViewDataEntities())
                {
                    // Used SQL statements because I didn't want to add more tables to the Entity
                    // Framework model just for this.  Also, it's easier.
                    db.Database.ExecuteSqlCommand("TRUNCATE TABLE dbo.Business");
                    db.Database.ExecuteSqlCommand("TRUNCATE TABLE dbo.BusinessReset");
                    db.Database.ExecuteSqlCommand("TRUNCATE TABLE dbo.Park");
                    db.Database.ExecuteSqlCommand("TRUNCATE TABLE dbo.ParkReset");
                    db.Database.ExecuteSqlCommand("TRUNCATE TABLE dbo.PublicFacility");
                    db.Database.ExecuteSqlCommand("TRUNCATE TABLE dbo.PublicFacilityReset");

                    db.Database.ExecuteSqlCommand("INSERT INTO dbo.BusinessReset SELECT " +
                        "TOP 50 * FROM dbo.BusinessBase ORDER BY NEWID()");
                    db.Database.ExecuteSqlCommand("INSERT INTO dbo.Business SELECT " +
                        "* FROM dbo.BusinessReset");
                    db.Database.ExecuteSqlCommand("INSERT INTO dbo.ParkReset SELECT " +
                        "TOP 50 * FROM dbo.ParkBase ORDER BY NEWID()");
                    db.Database.ExecuteSqlCommand("INSERT INTO dbo.Park SELECT " +
                        "* FROM dbo.ParkReset");
                    db.Database.ExecuteSqlCommand("INSERT INTO dbo.PublicFacilityReset SELECT " +
                        "TOP 50 * FROM dbo.PublicFacilityBase ORDER BY NEWID()");
                    db.Database.ExecuteSqlCommand("INSERT INTO dbo.PublicFacility SELECT " +
                        "* FROM dbo.PublicFacilityReset");
                }

                lblResult.Text = "The tables have been randomized.";
                lblResult.Visible = true;
            }
```

```
            catch
            {
                lblError.Text = "Error: Could not randomize the tables.";
                lblError.Visible = true;
            }
        }

        /*-------------------------------------------------------------------------------------
         * Name:    btnReset_Click
         * Type:    Event Handler Method
         * Purpose: Handles resetting the tables to their prior randomized states.
         * Input:   object sender, holds a reference to the object that raised this event.
         * Input:   EventArgs e, holds data related to this event.
         * Output:  Nothing.
        -------------------------------------------------------------------------------------*/
        protected void btnReset_Click(object sender, EventArgs e)
        {
            // Hide things until needed.
            lblResult.Visible = false;
            lblError.Visible = false;

            try
            {
                using (CViewDataEntities db = new CViewDataEntities())
                {
                    // Again, used SQL statements because I didn't want to add more tables to the
                    // Entity Framework model just for this.  Also, it's easier.
                    db.Database.ExecuteSqlCommand("TRUNCATE TABLE dbo.Business");
                    db.Database.ExecuteSqlCommand("TRUNCATE TABLE dbo.Park");
                    db.Database.ExecuteSqlCommand("TRUNCATE TABLE dbo.PublicFacility");

                    db.Database.ExecuteSqlCommand("INSERT INTO dbo.Business SELECT " +
                        "* FROM dbo.BusinessReset");
                    db.Database.ExecuteSqlCommand("INSERT INTO dbo.Park SELECT " +
                        "* FROM dbo.ParkReset");
                    db.Database.ExecuteSqlCommand("INSERT INTO dbo.PublicFacility SELECT " +
                        "* FROM dbo.PublicFacilityReset");
                }

                lblResult.Text = "The tables have been reset to their prior randomized states.";
                lblResult.Visible = true;
            }
            catch
            {
                lblError.Text = "Error: Could not reset the tables.";
                lblError.Visible = true;
            }
        }
    }
}
```

```
/*-----------------------------------------------------------------------------------------
 * Author:      Dan Cassidy
 * Date:        2015-06-23
 * Assignment:  cView-P4
 * Source File: Add.aspx.cs
 * Language:    C#
 * Course:      CSCI-C 490, C# Programming, MoWe 08:00
 * Purpose:     Code-behind file for Add.aspx. Controls the process of adding an item to the
 *              database via the Entity Framework model.
-----------------------------------------------------------------------------------------*/

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace cView_P4_DanCassidy
{
    public partial class Add : System.Web.UI.Page
    {
        /*-----------------------------------------------------------------------------------------
         * Name:    btnAdd_Click
         * Type:    Event Handler Method
         * Purpose: Handles constructing, errorchecking, and finally adding an object to the
         *          database.
         * Input:   object sender, holds a reference to the object that raised this event.
         * Input:   EventArgs e, holds data related to this event.
         * Output:  Nothing.
        -----------------------------------------------------------------------------------------*/
        protected void btnAdd_Click(object sender, EventArgs e)
        {
            object toAdd = null;

            switch ((Global.Enums.ItemTypes)ddlItemType.SelectedIndex)
            {
                case Global.Enums.ItemTypes.Business:
                    toAdd = new Business()
                    {
                        // Common Fields
                        Name = txtName.Text.Trim(),
                        Type = txtType.Text.Trim(),
                        StreetAddress = txtStreetAddress.Text.Trim(),
                        City = txtCity.Text.Trim(),
                        State = txtState.Text.Trim(),
                        Zip = txtZip.Text.Trim(),
                        Latitude = SimpleConvert.ToDecimal(txtLatitude.Text.Trim()),
                        Longitude = SimpleConvert.ToDecimal(txtLongitude.Text.Trim()),
                        Phone = txtPhone.Text.Trim(),

                        // Business Fields
                        LicenseNumber = txtLicenseNumber.Text.Trim(),
                        LicenseIssueDate = SimpleConvert.ToDateTime(
                            txtLicenseExpirDate.Text.Trim()),
                        LicenseExpirDate = SimpleConvert.ToDateTime(
                            txtLicenseExpirDate.Text.Trim()),
                        LicenseStatus = txtLicenseStatus.Text.Trim(),
                        CouncilDistrict = txtCouncilDistrict.Text.Trim()
                    };
                    break;

                case Global.Enums.ItemTypes.Park:
                    toAdd = new Park()
                    {
                        // Common Fields
```

```
                        Name = txtName.Text.Trim(),
                        Type = txtType.Text.Trim(),
                        StreetAddress = txtStreetAddress.Text.Trim(),
                        City = txtCity.Text.Trim(),
                        State = txtState.Text.Trim(),
                        Zip = txtZip.Text.Trim(),
                        Latitude = SimpleConvert.ToDecimal(txtLatitude.Text.Trim()),
                        Longitude = SimpleConvert.ToDecimal(txtLongitude.Text.Trim()),
                        Phone = txtPhone.Text.Trim(),

                        // Park Fields
                        FeatureBaseball = SimpleConvert.ToByte(txtFeatureBaseball.Text.Trim()),
                        FeatureBasketball = SimpleConvert.ToDecimal(
                            txtFeatureBasketball.Text.Trim()),
                        FeatureGolf = SimpleConvert.ToDecimal(txtFeatureGolf.Text.Trim()),
                        FeatureLargeMPField = SimpleConvert.ToByte(
                            txtFeatureLargeMPField.Text.Trim()),
                        FeatureTennis = SimpleConvert.ToByte(txtFeatureTennis.Text.Trim()),
                        FeatureVolleyball = SimpleConvert.ToByte(txtFeatureVolleyball.Text.Trim())
                    };
                    break;

                case Global.Enums.ItemTypes.PublicFacility:
                    toAdd = new PublicFacility()
                    {
                        // Common Fields
                        Name = txtName.Text.Trim(),
                        Type = txtType.Text.Trim(),
                        StreetAddress = txtStreetAddress.Text.Trim(),
                        City = txtCity.Text.Trim(),
                        State = txtState.Text.Trim(),
                        Zip = txtZip.Text.Trim(),
                        Latitude = SimpleConvert.ToDecimal(txtLatitude.Text.Trim()),
                        Longitude = SimpleConvert.ToDecimal(txtLongitude.Text.Trim()),
                        Phone = txtPhone.Text.Trim()
                    };
                    break;

                default:
                    // ... How?
                    return;
            }

            // Add the object to the database.
            if (toAdd != null)
            {
                try
                {
                    using (CViewDataEntities database = new CViewDataEntities())
                    {
                        if (toAdd is Business)
                        {
                            Business businessToAdd = toAdd as Business;

                            // Do error-checking.
                            if (businessToAdd.LicenseNumber == string.Empty ||
                                businessToAdd.LicenseNumber == null)
                            {
                                throw new Global.Exceptions.EmptyOrNullPKException(
                                    Global.Strings.BusinessKey);
                            }
                            else if (database.Businesses.Find(businessToAdd.LicenseNumber) != null)
                            {
                                throw new Global.Exceptions.DuplicatePKException(
                                    Global.Strings.BusinessKey, businessToAdd.LicenseNumber);
                            }
```

```
                            // If everything is ok, add item to table.
                            database.Businesses.Add(businessToAdd);
                        }
                        else if (toAdd is Park)
                        {
                            Park parkToAdd = toAdd as Park;

                            //Do error-checking.
                            if (parkToAdd.Name == string.Empty ||
                                parkToAdd.Name == null)
                            {
                                throw new Global.Exceptions.EmptyOrNullPKException(
                                    Global.Strings.ParkKey);
                            }
                            else if (database.Parks.Find(parkToAdd.Name) != null)
                            {
                                throw new Global.Exceptions.DuplicatePKException(
                                    Global.Strings.ParkKey, parkToAdd.Name);
                            }

                            // If everything is ok, add item to table.
                            database.Parks.Add(parkToAdd);
                        }
                        else if (toAdd is PublicFacility)
                        {
                            PublicFacility publicFacilityToAdd = toAdd as PublicFacility;

                            //Do error-checking.
                            if (publicFacilityToAdd.Name == string.Empty ||
                                publicFacilityToAdd.Name == null)
                            {
                                throw new Global.Exceptions.EmptyOrNullPKException(
                                    Global.Strings.PublicFacilityKey);
                            }
                            else if (database.PublicFacilities.Find(publicFacilityToAdd.Name) !=
                                     null)
                            {
                                throw new Global.Exceptions.DuplicatePKException(
                                    Global.Strings.PublicFacilityKey, publicFacilityToAdd.Name);
                            }

                            // If everything is ok, add item to table.
                            database.PublicFacilities.Add(publicFacilityToAdd);
                        }

                        database.SaveChanges();

                        lblResult.Text = "Added record to the table.";
                        lblResult.Visible = true;
                        lblError.Visible = false;
                    }
                }
                catch (Exception ex)
                {
                    // Drill down to the innermost exception.
                    while (ex.InnerException != null)
                        ex = ex.InnerException;

                    lblError.Text = "Error: " + ex.Message;
                    lblError.Visible = true;
                    lblResult.Visible = false;
                }
            }
        }

```

```
        /*-------------------------------------------------------------------------------------
         * Name:    ddlItemType_SelectedIndexChanged
         * Type:    Event Handler Method
         * Purpose: Handles showing and hiding the various controls to allow a user to enter
         *          information so that an item can be added to the database.
         * Input:   object sender, holds a reference to the object that raised this event.
         * Input:   EventArgs e, holds data related to this event.
         * Output:  Nothing.
        -------------------------------------------------------------------------------------*/
        protected void ddlItemType_SelectedIndexChanged(object sender, EventArgs e)
        {
            // Hide things until needed.
            lblError.Visible = false;
            lblResult.Visible = false;

            mViewBasic.ActiveViewIndex = -1;
            mViewSpecific.ActiveViewIndex = -1;
            btnAdd.Visible = false;

            // Decide which controls should be shown to enable user to add an item.
            switch ((Global.Enums.ItemTypes)ddlItemType.SelectedIndex)
            {
                case Global.Enums.ItemTypes.Business:
                    mViewBasic.ActiveViewIndex = 0;
                    mViewSpecific.ActiveViewIndex = 0;
                    lblName.Text = Global.Strings.BusinessName + Global.Strings.Separator;
                    lblType.Text = Global.Strings.BusinessType + Global.Strings.Separator;
                    btnAdd.Visible = true;
                    break;

                case Global.Enums.ItemTypes.Park:
                    mViewBasic.ActiveViewIndex = 0;
                    mViewSpecific.ActiveViewIndex = 1;
                    lblName.Text = Global.Strings.ParkName + Global.Strings.Separator;
                    lblType.Text = Global.Strings.ParkType + Global.Strings.Separator;
                    btnAdd.Visible = true;
                    break;

                case Global.Enums.ItemTypes.PublicFacility:
                    mViewBasic.ActiveViewIndex = 0;
                    mViewSpecific.ActiveViewIndex = -1;
                    lblName.Text = Global.Strings.PublicFacilityName + Global.Strings.Separator;
                    lblType.Text = Global.Strings.PublicFacilityType + Global.Strings.Separator;
                    btnAdd.Visible = true;
                    break;

                default:
                    break;
            }
        }
    }
}
```

```
/*-----------------------------------------------------------------------------------------------
 * Author:      Dan Cassidy
 * Date:        2015-06-23
 * Assignment:  cView-P4
 * Source File: Modify.aspx.cs
 * Language:    C#
 * Course:      CSCI-C 490, C# Programming, MoWe 08:00
 * Purpose:     Code-behind file for Modify.aspx.  Controls the process of modifying an item in the
 *              database via the Entity Framework model.
-----------------------------------------------------------------------------------------------*/

using System;
using System.Collections.Generic;
using System.Data.Entity;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace cView_P4_DanCassidy
{
    public partial class Modify : System.Web.UI.Page
    {
        /*-----------------------------------------------------------------------------------------
         * Name:    btnModify_Click
         * Type:    Event Handler Method
         * Purpose: Handles showing and hiding the various controls to aid in allowing the user to
         *          change the data associated with the chosen item.
         * Input:   object sender, holds a reference to the object that raised this event.
         * Input:   EventArgs e, holds data related to this event.
         * Output:  Nothing.
        -----------------------------------------------------------------------------------------*/
        protected void btnModify_Click(object sender, EventArgs e)
        {
            // Quick check to make sure that something is selected.
            switch ((Global.Enums.ItemTypes)ddlItemType.SelectedIndex)
            {
                case Global.Enums.ItemTypes.Business:
                    if (gViewBusiness.SelectedIndex == -1)
                        return;
                    break;

                case Global.Enums.ItemTypes.Park:
                    if (gViewPark.SelectedIndex == -1)
                        return;
                    break;

                case Global.Enums.ItemTypes.PublicFacility:
                    if (gViewPublicFacility.SelectedIndex == -1)
                        return;
                    break;

                default:
                    break;
            }

            // Hide things until needed.
            lblError.Visible = false;
            lblResult.Visible = false;
            mViewDisplay.ActiveViewIndex = -1;
            mViewModifyBasic.ActiveViewIndex = -1;
            mViewModifySpecific.ActiveViewIndex = -1;
            btnModify.Visible = false;
            btnSaveChanges.Visible = false;
            btnBack.Visible = false;

```

```
            try
            {
                using (CViewDataEntities database = new CViewDataEntities())
                {
                    object keyToModify = null;

                    // Choose what to display based on the selected item type.
                    switch ((Global.Enums.ItemTypes)ddlItemType.SelectedIndex)
                    {
                        case Global.Enums.ItemTypes.Business:
                            // Basic UI prep.
                            mViewModifyBasic.ActiveViewIndex = 0;
                            mViewModifySpecific.ActiveViewIndex = 0;
                            lblName.Text = Global.Strings.BusinessName + Global.Strings.Separator;
                            lblType.Text = Global.Strings.BusinessType + Global.Strings.Separator;
                            btnSaveChanges.Visible = true;
                            btnBack.Visible = true;

                            // Get object to modify.
                            keyToModify = gViewBusiness.SelectedDataKey.Value;
                            Business businessToModify = database.Businesses.Find(keyToModify);

                            // Fill in textbox values.
                            txtName.Text = businessToModify.Name;
                            txtName.Enabled = true;
                            txtType.Text = businessToModify.Type;
                            txtStreetAddress.Text = businessToModify.StreetAddress;
                            txtCity.Text = businessToModify.City;
                            txtState.Text = businessToModify.State;
                            txtZip.Text = businessToModify.Zip;
                            txtLatitude.Text = businessToModify.Latitude.ToString();
                            txtLongitude.Text = businessToModify.Longitude.ToString();
                            txtPhone.Text = businessToModify.Phone;
                            txtLicenseNumber.Text = businessToModify.LicenseNumber;
                            txtLicenseIssueDate.Text = string.Format("{0:d}",
                                businessToModify.LicenseIssueDate);
                            txtLicenseExpirDate.Text = string.Format("{0:d}",
                                businessToModify.LicenseExpirDate);
                            txtLicenseStatus.Text = businessToModify.LicenseStatus;
                            txtCouncilDistrict.Text = businessToModify.CouncilDistrict;
                            break;

                        case Global.Enums.ItemTypes.Park:
                            // Basic UI prep.
                            mViewModifyBasic.ActiveViewIndex = 0;
                            mViewModifySpecific.ActiveViewIndex = 1;
                            lblName.Text = Global.Strings.ParkName + Global.Strings.Separator;
                            lblType.Text = Global.Strings.ParkType + Global.Strings.Separator;
                            btnSaveChanges.Visible = true;
                            btnBack.Visible = true;

                            // Get object to modify.
                            keyToModify = gViewPark.SelectedDataKey.Value;
                            Park parkToModify = database.Parks.Find(keyToModify);

                            // Fill in textbox values.
                            txtName.Text = parkToModify.Name;
                            txtName.Enabled = false;
                            txtType.Text = parkToModify.Type;
                            txtStreetAddress.Text = parkToModify.StreetAddress;
                            txtCity.Text = parkToModify.City;
                            txtState.Text = parkToModify.State;
                            txtZip.Text = parkToModify.Zip;
                            txtLatitude.Text = parkToModify.Latitude.ToString();
                            txtLongitude.Text = parkToModify.Longitude.ToString();
                            txtPhone.Text = parkToModify.Phone;
```

```
                            txtFeatureBaseball.Text = parkToModify.FeatureBaseball.ToString();
                            txtFeatureBasketball.Text = parkToModify.FeatureBasketball.ToString();
                            txtFeatureGolf.Text = parkToModify.FeatureGolf.ToString();
                            txtFeatureLargeMPField.Text = parkToModify.FeatureLargeMPField.
                                ToString();
                            txtFeatureTennis.Text = parkToModify.FeatureTennis.ToString();
                            txtFeatureVolleyball.Text = parkToModify.FeatureVolleyball.ToString();
                            break;

                        case Global.Enums.ItemTypes.PublicFacility:
                            // Basic UI prep.
                            mViewModifyBasic.ActiveViewIndex = 0;
                            lblName.Text = Global.Strings.PublicFacilityName +
                                Global.Strings.Separator;
                            lblType.Text = Global.Strings.PublicFacilityType +
                                Global.Strings.Separator;
                            btnSaveChanges.Visible = true;
                            btnBack.Visible = true;

                            // Get object to modify.
                            keyToModify = gViewPublicFacility.SelectedDataKey.Value;
                            PublicFacility publicFacilityToModify = database.PublicFacilities.
                                Find(keyToModify);

                            // Fill in textbox values.
                            txtName.Text = publicFacilityToModify.Name;
                            txtName.Enabled = false;
                            txtType.Text = publicFacilityToModify.Type;
                            txtStreetAddress.Text = publicFacilityToModify.StreetAddress;
                            txtCity.Text = publicFacilityToModify.City;
                            txtState.Text = publicFacilityToModify.State;
                            txtZip.Text = publicFacilityToModify.Zip;
                            txtLatitude.Text = publicFacilityToModify.Latitude.ToString();
                            txtLongitude.Text = publicFacilityToModify.Longitude.ToString();
                            txtPhone.Text = publicFacilityToModify.Phone;
                            break;

                        default:
                            throw new InvalidOperationException(
                                "Invalid item type dropdown value.");
                    }
                }
            }
            catch (Exception ex)
            {
                lblError.Text = "Error: " + ex.Message;
                lblError.Visible = true;
            }
        }

        /*-----------------------------------------------------------------------------------------
         * Name:    btnSaveChanges_Click
         * Type:    Event Handler Method
         * Purpose: Handles saving the changed data for the specific item back to the database via
         *          the Entity Framework model.
         * Input:   object sender, holds a reference to the object that raised this event.
         * Input:   EventArgs e, holds data related to this event.
         * Output:  Nothing.
        -----------------------------------------------------------------------------------------*/
        protected void btnSaveChanges_Click(object sender, EventArgs e)
        {
            // Hide things until needed.
            lblError.Visible = false;
            lblResult.Visible = false;
            mViewDisplay.ActiveViewIndex = -1;
            mViewModifyBasic.ActiveViewIndex = -1;
```

```
            mViewModifySpecific.ActiveViewIndex = -1;
            btnModify.Visible = false;
            btnSaveChanges.Visible = false;
            btnBack.Visible = false;

            try
            {
                using (CViewDataEntities database = new CViewDataEntities())
                {
                    object keyToModify = null;

                    // Choose what to do based on the selected item type.
                    switch ((Global.Enums.ItemTypes)ddlItemType.SelectedIndex)
                    {
                        case Global.Enums.ItemTypes.Business:
                            // Get object to modify.
                            keyToModify = gViewBusiness.SelectedDataKey.Value;
                            Business businessToModify = database.Businesses.Find(keyToModify);

                            // Save values to object.
                            businessToModify.Name = txtName.Text.Trim();
                            businessToModify.Type = txtType.Text.Trim();
                            businessToModify.StreetAddress = txtStreetAddress.Text.Trim();
                            businessToModify.City = txtCity.Text.Trim();
                            businessToModify.State = txtState.Text.Trim();
                            businessToModify.Zip = txtZip.Text.Trim();
                            businessToModify.Latitude = SimpleConvert.ToDecimal(txtLatitude.Text.
                                Trim());
                            businessToModify.Longitude = SimpleConvert.ToDecimal(txtLongitude.Text.
                                Trim());
                            businessToModify.Phone = txtPhone.Text.Trim();
                            businessToModify.LicenseIssueDate = SimpleConvert.ToDateTime(
                                txtLicenseIssueDate.Text.Trim());
                            businessToModify.LicenseExpirDate = SimpleConvert.ToDateTime(
                                txtLicenseExpirDate.Text.Trim());
                            businessToModify.LicenseStatus = txtLicenseStatus.Text.Trim();
                            businessToModify.CouncilDistrict = txtCouncilDistrict.Text.Trim();

                            // Modelled off http://stackoverflow.com/a/15339512.
                            database.Businesses.Attach(businessToModify);
                            database.Entry(businessToModify).State = EntityState.Modified;
                            break;

                        case Global.Enums.ItemTypes.Park:
                            // Get object to modify.
                            keyToModify = gViewPark.SelectedDataKey.Value;
                            Park parkToModify = database.Parks.Find(keyToModify);

                            // Save values to object.
                            parkToModify.Type = txtType.Text.Trim();
                            parkToModify.StreetAddress = txtStreetAddress.Text.Trim();
                            parkToModify.City = txtCity.Text.Trim();
                            parkToModify.State = txtState.Text.Trim();
                            parkToModify.Zip = txtZip.Text.Trim();
                            parkToModify.Latitude = SimpleConvert.ToDecimal(txtLatitude.Text.
                                Trim());
                            parkToModify.Longitude = SimpleConvert.ToDecimal(txtLongitude.Text.
                                Trim());
                            parkToModify.Phone = txtPhone.Text.Trim();
                            parkToModify.FeatureBaseball = SimpleConvert.ToByte(
                                txtFeatureBaseball.Text.Trim());
                            parkToModify.FeatureBasketball = SimpleConvert.ToDecimal(
                                txtFeatureBasketball.Text.Trim());
                            parkToModify.FeatureGolf = SimpleConvert.ToDecimal(
                                txtFeatureGolf.Text.Trim());
                            parkToModify.FeatureLargeMPField = SimpleConvert.ToByte(
```

```
                                txtFeatureLargeMPField.Text.Trim());
                            parkToModify.FeatureTennis = SimpleConvert.ToByte(
                                txtFeatureTennis.Text.Trim());
                            parkToModify.FeatureVolleyball = SimpleConvert.ToByte(
                                txtFeatureVolleyball.Text.Trim());

                            // Modelled off http://stackoverflow.com/a/15339512.
                            database.Parks.Attach(parkToModify);
                            database.Entry(parkToModify).State = EntityState.Modified;
                            break;

                        case Global.Enums.ItemTypes.PublicFacility:
                            // Get object to modify.
                            keyToModify = gViewPublicFacility.SelectedDataKey.Value;
                            PublicFacility publicFacilityToModify = database.PublicFacilities.
                                Find(keyToModify);

                            // Save values to object.
                            publicFacilityToModify.Type = txtType.Text.Trim();
                            publicFacilityToModify.StreetAddress = txtStreetAddress.Text.Trim();
                            publicFacilityToModify.City = txtCity.Text.Trim();
                            publicFacilityToModify.State = txtState.Text.Trim();
                            publicFacilityToModify.Zip = txtZip.Text.Trim();
                            publicFacilityToModify.Latitude = SimpleConvert.ToDecimal(
                                txtLatitude.Text.Trim());
                            publicFacilityToModify.Longitude = SimpleConvert.ToDecimal(
                                txtLongitude.Text.Trim());
                            publicFacilityToModify.Phone = txtPhone.Text.Trim();

                            // Modelled off http://stackoverflow.com/a/15339512.
                            database.PublicFacilities.Attach(publicFacilityToModify);
                            database.Entry(publicFacilityToModify).State = EntityState.Modified;
                            break;

                        default:
                            throw new InvalidOperationException(
                                "Invalid item type dropdown value.");
                    }
                    // Save any changes to the database and refresh gridviews.
                    database.SaveChanges();
                    mViewDisplay.DataBind();
                }
                // Go back to item selection and let the user know the operation was successful.
                ddlItemType_SelectedIndexChanged(sender, e);
                lblResult.Text = "Item modified successfully.";
                lblResult.Visible = true;
            }
            catch (Exception ex)
            {
                lblError.Text = "Error: " + ex.Message;
                lblError.Visible = true;
            }
        }

        /*-------------------------------------------------------------------------------------
         * Name:    ddlItemType_SelectedIndexChanged
         * Type:    Event Handler Method
         * Purpose: Handles showing and hiding the various controls to aid in allowing the user to
         *          select an item to modify.
         * Input:   object sender, holds a reference to the object that raised this event.
         * Input:   EventArgs e, holds data related to this event.
         * Output:  Nothing.
        -------------------------------------------------------------------------------------*/
        protected void ddlItemType_SelectedIndexChanged(object sender, EventArgs e)
        {
            // Hide things until needed.
```

```
            lblError.Visible = false;
            lblResult.Visible = false;
            mViewDisplay.ActiveViewIndex = -1;
            mViewModifyBasic.ActiveViewIndex = -1;
            mViewModifySpecific.ActiveViewIndex = -1;
            btnModify.Visible = false;
            btnSaveChanges.Visible = false;
            btnBack.Visible = false;

            // Reset selected indexes of GridView controls.
            gViewBusiness.SelectedIndex = -1;
            gViewPark.SelectedIndex = -1;
            gViewPublicFacility.SelectedIndex = -1;

            // Show the appropriate view.
            switch ((Global.Enums.ItemTypes)ddlItemType.SelectedIndex)
            {
                case Global.Enums.ItemTypes.Business:
                    mViewDisplay.ActiveViewIndex = 0;
                    btnModify.Visible = true;
                    break;

                case Global.Enums.ItemTypes.Park:
                    mViewDisplay.ActiveViewIndex = 1;
                    btnModify.Visible = true;
                    break;

                case Global.Enums.ItemTypes.PublicFacility:
                    mViewDisplay.ActiveViewIndex = 2;
                    btnModify.Visible = true;
                    break;

                default:
                    break;
            }
        }
    }
}
```

```
/*-----------------------------------------------------------------------------------------------
 * Author:      Dan Cassidy
 * Date:        2015-06-23
 * Assignment:  cView-P4
 * Source File: Delete.aspx.cs
 * Language:    C#
 * Course:      CSCI-C 490, C# Programming, MoWe 08:00
 * Purpose:     Code-behind file for Delete.aspx.  Controls the deletion of a single row of data (1
 *              object) from a table.
-----------------------------------------------------------------------------------------------*/

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace cView_P4_DanCassidy
{
    public partial class Delete : System.Web.UI.Page
    {
        /*-----------------------------------------------------------------------------------------
         * Name:    btnDelete_Click
         * Type:    Event Handler Method
         * Purpose: Handles deleting an object from the database.
         * Input:   object sender, holds a reference to the object that raised this event.
         * Input:   EventArgs e, holds data related to this event.
         * Output:  Nothing.
        -----------------------------------------------------------------------------------------*/
        protected void btnDelete_Click(object sender, EventArgs e)
        {
            // Dummy polymorphoc object.
            object keyToDelete = null;

            // Attempt to delete a record.
            try
            {
                using (CViewDataEntities database = new CViewDataEntities())
                {
                    switch ((Global.Enums.ItemTypes)ddlItemType.SelectedIndex)
                    {
                        case Global.Enums.ItemTypes.Business:
                            if (gViewBusiness.SelectedIndex == -1)
                                return;
                            keyToDelete = gViewBusiness.SelectedDataKey.Value;
                            gViewBusiness.SelectRow(-1);
                            Business businessToDelete = database.Businesses.Find(keyToDelete);
                            database.Businesses.Remove(businessToDelete);
                            break;

                        case Global.Enums.ItemTypes.Park:
                            if (gViewPark.SelectedIndex == -1)
                                return;
                            keyToDelete = gViewPark.SelectedDataKey.Value;
                            gViewPark.SelectRow(-1);
                            Park parkToDelete = database.Parks.Find(keyToDelete);
                            database.Parks.Remove(parkToDelete);
                            break;

                        case Global.Enums.ItemTypes.PublicFacility:
                            if (gViewPublicFacility.SelectedIndex == -1)
                                return;
                            keyToDelete = gViewPublicFacility.SelectedDataKey.Value;
                            gViewPublicFacility.SelectRow(-1);
                            PublicFacility publicFacilityToDelete =
```

```
                                database.PublicFacilities.Find(keyToDelete);
                            database.PublicFacilities.Remove(publicFacilityToDelete);
                            break;

                        default:
                            // ... How?
                            return;
                    }

                    database.SaveChanges();
                    mViewData.DataBind();
                }

                lblResult.Text = "Row successfully deleted.";
                lblResult.Visible = true;
                lblError.Visible = false;
            }
            catch
            {
                lblError.Text = "Error: Could not delete the specified row.";
                lblError.Visible = true;
                lblResult.Visible = false;
            }
        }

        /*-----------------------------------------------------------------------------------------
         * Name:    ddlItemType_SelectedIndexChanged
         * Type:    Event Handler Method
         * Purpose: Handles showing and hiding the various controls to allow a user to choose an
         *          item to be deleted from the database.
         * Input:   object sender, holds a reference to the object that raised this event.
         * Input:   EventArgs e, holds data related to this event.
         * Output:  Nothing.
        -----------------------------------------------------------------------------------------*/
        protected void ddlItemType_SelectedIndexChanged(object sender, EventArgs e)
        {
            // Hide things until needed.
            lblError.Visible = false;
            lblResult.Visible = false;

            mViewData.ActiveViewIndex = -1;
            btnDelete.Visible = false;

            // Decide which controls should be shown to enable user to delete an item.
            switch ((Global.Enums.ItemTypes)ddlItemType.SelectedIndex)
            {
                case Global.Enums.ItemTypes.Business:
                    mViewData.ActiveViewIndex = 0;
                    btnDelete.Visible = true;
                    break;

                case Global.Enums.ItemTypes.Park:
                    mViewData.ActiveViewIndex = 1;
                    btnDelete.Visible = true;
                    break;

                case Global.Enums.ItemTypes.PublicFacility:
                    mViewData.ActiveViewIndex = 2;
                    btnDelete.Visible = true;
                    break;

                default:
                    break;
            }
        }

```

```
    }
}
```

```
/*-----------------------------------------------------------------------------------------------
 * Author:      Dan Cassidy
 * Date:        2015-06-23
 * Assignment:  cView-P4
 * Source File: Display.aspx.cs
 * Language:    C#
 * Course:      CSCI-C 490, C# Programming, MoWe 08:00
 * Purpose:     Code-behind file for Display.aspx.  Controls the display of the different tables of
 *              items.
-----------------------------------------------------------------------------------------------*/

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace cView_P4_DanCassidy
{
    public partial class Display : System.Web.UI.Page
    {
        /*-----------------------------------------------------------------------------------------
         * Name:    ddlItemType_SelectedIndexChanged
         * Type:    Event Handler Method
         * Purpose: Handles showing and hiding the data views.
         * Input:   object sender, holds a reference to the object that raised this event.
         * Input:   EventArgs e, holds data related to this event.
         * Output:  Nothing.
        -----------------------------------------------------------------------------------------*/
        protected void ddlItemType_SelectedIndexChanged(object sender, EventArgs e)
        {
            switch ((Global.Enums.ItemTypes)ddlItemType.SelectedIndex)
            {
                case Global.Enums.ItemTypes.Business:
                    mViewData.ActiveViewIndex = 0;
                    break;

                case Global.Enums.ItemTypes.Park:
                    mViewData.ActiveViewIndex = 1;
                    break;

                case Global.Enums.ItemTypes.PublicFacility:
                    mViewData.ActiveViewIndex = 2;
                    break;

                default:
                    mViewData.ActiveViewIndex = -1;
                    break;
            }
        }
    }
}
```

```
/*-----------------------------------------------------------------------------------------------
 * Author:      Dan Cassidy
 * Date:        2015-06-23
 * Assignment:  cView-P4
 * Source File: Search.aspx.cs
 * Language:    C#
 * Course:      CSCI-C 490, C# Programming, MoWe 08:00
 * Purpose:     Code-behind file for Search.aspx. Controls the interface whereby a user can search
 *              the tables.
-----------------------------------------------------------------------------------------------*/

using System;
using System.Collections.Generic;
using System.Data.Entity;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace cView_P4_DanCassidy
{
    public partial class Search : System.Web.UI.Page
    {
        /*-----------------------------------------------------------------------------------------
         * Name:    btnSearch_Click
         * Type:    Event Handler Method
         * Purpose: Handles the actual search and display of the results.
         * Input:   object sender, holds a reference to the object that raised this event.
         * Input:   EventArgs e, holds data related to this event.
         * Output:  Nothing.
        -----------------------------------------------------------------------------------------*/
        protected void btnSearch_Click(object sender, EventArgs e)
        {
            // Hide things until needed.
            lblError.Visible = false;
            lblResult.Visible = false;
            mViewSearchResults.ActiveViewIndex = -1;

            // Pre-conversions to save some space.
            string toSearchFor = txtSearch.Text.Trim();
            byte toSearchForByte = SimpleConvert.ToByte(toSearchFor);
            DateTime toSearchForDateTime = SimpleConvert.ToDateTime(toSearchFor);
            decimal toSearchForDecimal = SimpleConvert.ToDecimal(toSearchFor);

            int resultCount = 0;
            int viewToDisplay = -1;
            GridView gViewToDisplay;

            int selectedIndex;
            object baseObject;

            // Determine what comparator to use; default is "|" for "contains".
            string comparator = "|";
            if (ddlComparatorsStrings.Visible == true)
            {
                switch ((Global.Enums.ComparatorsStrings)ddlComparatorsStrings.SelectedIndex)
                {
                    case Global.Enums.ComparatorsStrings.NotContain:
                        comparator = "!|";
                        break;

                    case Global.Enums.ComparatorsStrings.Equal:
                        comparator = "==";
                        break;

                    case Global.Enums.ComparatorsStrings.NotEqual:
```

```
                        comparator = "!=";
                        break;

                    default:
                        break;
                }
            }
            else if (ddlComparatorsNotStrings.Visible == true)
            {
                switch ((Global.Enums.ComparatorsNotStrings)ddlComparatorsNotStrings.SelectedIndex)
                {
                    case Global.Enums.ComparatorsNotStrings.NotContain:
                        comparator = "!|";
                        break;

                    case Global.Enums.ComparatorsNotStrings.Equal:
                        comparator = "==";
                        break;

                    case Global.Enums.ComparatorsNotStrings.NotEqual:
                        comparator = "!=";
                        break;

                    case Global.Enums.ComparatorsNotStrings.Greater:
                        comparator = ">";
                        break;

                    case Global.Enums.ComparatorsNotStrings.Less:
                        comparator = "<";
                        break;

                    case Global.Enums.ComparatorsNotStrings.GreaterEqual:
                        comparator = ">=";
                        break;

                    case Global.Enums.ComparatorsNotStrings.LessEqual:
                        comparator = "<=";
                        break;

                    default:
                        break;
                }
            }

            // Do the search.
            //
            // The method I ended up using is a bit kludgy, but I had to resort to this because LINQ
            // to Entities is stupid and doesn't even try to evaluate what it can server-side prior
            // to attempting to translate the query to SQL and failing because indexers are not a
            // SQL thing.
            try
            {
                using (CViewDataEntities database = new CViewDataEntities())
                {
                    switch ((Global.Enums.ItemTypes)ddlItemType.SelectedIndex)
                    {
                        case Global.Enums.ItemTypes.Business:
                            viewToDisplay = 0;
                            gViewToDisplay = gViewBusinessResults;
                            selectedIndex = ddlBusiness.SelectedIndex;
                            baseObject = database.Businesses.First()[selectedIndex];
                            if (baseObject != null)
                            {
                                IEnumerable<Business> searchResults = null;
                                DbSet<Business> tableToSearch = database.Businesses;
                                switch (comparator)
```

```
                                {
                                    case "|":
                                        searchResults = tableToSearch.AsEnumerable().Where(i =>
                                            i[selectedIndex].ToString().IndexOf(
                                            toSearchFor, StringComparison.OrdinalIgnoreCase) >= 0);
                                        break;

                                    case "!|":
                                        searchResults = tableToSearch.AsEnumerable().Where(i =>
                                            i[selectedIndex].ToString().IndexOf(
                                            toSearchFor, StringComparison.OrdinalIgnoreCase) < 0);
                                        break;

                                    case "==":
                                        if (baseObject is byte)
                                            searchResults = tableToSearch.AsEnumerable().Where(i =>
                                                (byte)i[selectedIndex] ==
                                                toSearchForByte);
                                        else if (baseObject is DateTime)
                                            searchResults = tableToSearch.AsEnumerable().Where(i =>
                                                (DateTime)i[selectedIndex] ==
                                                toSearchForDateTime);
                                        else if (baseObject is decimal)
                                            searchResults = tableToSearch.AsEnumerable().Where(i =>
                                                (decimal)i[selectedIndex] ==
                                                toSearchForDecimal);
                                        else
                                            searchResults = tableToSearch.AsEnumerable().Where(i =>
                                                ((string)i[selectedIndex]).ToLower() == toSearchFor.
                                                ToLower());
                                        break;

                                    case "!=":
                                        if (baseObject is byte)
                                            searchResults = tableToSearch.AsEnumerable().Where(i =>
                                                (byte)i[selectedIndex] !=
                                                toSearchForByte);
                                        else if (baseObject is DateTime)
                                            searchResults = tableToSearch.AsEnumerable().Where(i =>
                                                (DateTime)i[selectedIndex] !=
                                                toSearchForDateTime);
                                        else if (baseObject is decimal)
                                            searchResults = tableToSearch.AsEnumerable().Where(i =>
                                                (decimal)i[selectedIndex] !=
                                                toSearchForDecimal);
                                        else
                                            searchResults = tableToSearch.AsEnumerable().Where(i =>
                                                ((string)i[selectedIndex]).ToLower() != toSearchFor.
                                                ToLower());
                                        break;

                                    case ">":
                                        if (baseObject is byte)
                                            searchResults = tableToSearch.AsEnumerable().Where(i =>
                                                (byte)i[selectedIndex] >
                                                toSearchForByte);
                                        else if (baseObject is DateTime)
                                            searchResults = tableToSearch.AsEnumerable().Where(i =>
                                                (DateTime)i[selectedIndex] >
                                                toSearchForDateTime);
                                        else if (baseObject is decimal)
                                            searchResults = tableToSearch.AsEnumerable().Where(i =>
                                                (decimal)i[selectedIndex] >
                                                toSearchForDecimal);
                                        else
                                            throw new InvalidOperationException(
```

```
                                                "Comparator \">\" cannot be applied to strings.");
                                        break;

                                    case "<":
                                        if (baseObject is byte)
                                            searchResults = tableToSearch.AsEnumerable().Where(i =>
                                                (byte)i[selectedIndex] <
                                                toSearchForByte);
                                        else if (baseObject is DateTime)
                                            searchResults = tableToSearch.AsEnumerable().Where(i =>
                                                (DateTime)i[selectedIndex] <
                                                toSearchForDateTime);
                                        else if (baseObject is decimal)
                                            searchResults = tableToSearch.AsEnumerable().Where(i =>
                                                (decimal)i[selectedIndex] <
                                                toSearchForDecimal);
                                        else
                                            throw new InvalidOperationException(
                                                "Comparator \"<\" cannot be applied to strings.");
                                        break;

                                    case ">=":
                                        if (baseObject is byte)
                                            searchResults = tableToSearch.AsEnumerable().Where(i =>
                                                (byte)i[selectedIndex] >=
                                                toSearchForByte);
                                        else if (baseObject is DateTime)
                                            searchResults = tableToSearch.AsEnumerable().Where(i =>
                                                (DateTime)i[selectedIndex] >=
                                                toSearchForDateTime);
                                        else if (baseObject is decimal)
                                            searchResults = tableToSearch.AsEnumerable().Where(i =>
                                                (decimal)i[selectedIndex] >=
                                                toSearchForDecimal);
                                        else
                                            throw new InvalidOperationException(
                                                "Comparator \">=\" cannot be applied to strings.");
                                        break;

                                    case "<=":
                                        if (baseObject is byte)
                                            searchResults = tableToSearch.AsEnumerable().Where(i =>
                                                (byte)i[selectedIndex] <=
                                                toSearchForByte);
                                        else if (baseObject is DateTime)
                                            searchResults = tableToSearch.AsEnumerable().Where(i =>
                                                (DateTime)i[selectedIndex] <=
                                                toSearchForDateTime);
                                        else if (baseObject is decimal)
                                            searchResults = tableToSearch.AsEnumerable().Where(i =>
                                                (decimal)i[selectedIndex] <=
                                                toSearchForDecimal);
                                        else
                                            throw new InvalidOperationException(
                                                "Comparator \"<=\" cannot be applied to strings.");
                                        break;

                                    default:
                                        throw new InvalidOperationException("Invalid comparator.");
                                }
                                resultCount = searchResults.Count();
                                gViewToDisplay.DataSource = searchResults.ToList();
                            }
                            break;

                        case Global.Enums.ItemTypes.Park:
```

```
                            viewToDisplay = 1;
                            gViewToDisplay = gViewParkResults;
                            selectedIndex = ddlPark.SelectedIndex;
                            baseObject = database.Parks.First()[selectedIndex];
                            if (baseObject != null)
                            {
                                IEnumerable<Park> searchResults = null;
                                DbSet<Park> tableToSearch = database.Parks;
                                switch (comparator)
                                {
                                    case "|":
                                        searchResults = tableToSearch.AsEnumerable().Where(i =>
                                            i[selectedIndex].ToString().IndexOf(
                                            toSearchFor, StringComparison.OrdinalIgnoreCase) >= 0);
                                        break;

                                    case "!|":
                                        searchResults = tableToSearch.AsEnumerable().Where(i =>
                                            i[selectedIndex].ToString().IndexOf(
                                            toSearchFor, StringComparison.OrdinalIgnoreCase) < 0);
                                        break;

                                    case "==":
                                        if (baseObject is byte)
                                            searchResults = tableToSearch.AsEnumerable().Where(i =>
                                                (byte)i[selectedIndex] ==
                                                toSearchForByte);
                                        else if (baseObject is DateTime)
                                            searchResults = tableToSearch.AsEnumerable().Where(i =>
                                                (DateTime)i[selectedIndex] ==
                                                toSearchForDateTime);
                                        else if (baseObject is decimal)
                                            searchResults = tableToSearch.AsEnumerable().Where(i =>
                                                (decimal)i[selectedIndex] ==
                                                toSearchForDecimal);
                                        else
                                            searchResults = tableToSearch.AsEnumerable().Where(i =>
                                                ((string)i[selectedIndex]).ToLower() == toSearchFor.
                                                ToLower());
                                        break;

                                    case "!=":
                                        if (baseObject is byte)
                                            searchResults = tableToSearch.AsEnumerable().Where(i =>
                                                (byte)i[selectedIndex] !=
                                                toSearchForByte);
                                        else if (baseObject is DateTime)
                                            searchResults = tableToSearch.AsEnumerable().Where(i =>
                                                (DateTime)i[selectedIndex] !=
                                                toSearchForDateTime);
                                        else if (baseObject is decimal)
                                            searchResults = tableToSearch.AsEnumerable().Where(i =>
                                                (decimal)i[selectedIndex] !=
                                                toSearchForDecimal);
                                        else
                                            searchResults = tableToSearch.AsEnumerable().Where(i =>
                                                ((string)i[selectedIndex]).ToLower() != toSearchFor.
                                                ToLower());
                                        break;

                                    case ">":
                                        if (baseObject is byte)
                                            searchResults = tableToSearch.AsEnumerable().Where(i =>
                                                (byte)i[selectedIndex] >
                                                toSearchForByte);
                                        else if (baseObject is DateTime)
```

```
                                        searchResults = tableToSearch.AsEnumerable().Where(i =>
                                            (DateTime)i[selectedIndex] >
                                            toSearchForDateTime);
                                    else if (baseObject is decimal)
                                        searchResults = tableToSearch.AsEnumerable().Where(i =>
                                            (decimal)i[selectedIndex] >
                                            toSearchForDecimal);
                                    else
                                        throw new InvalidOperationException(
                                            "Comparator \">\" cannot be applied to strings.");
                                    break;

                                case "<":
                                    if (baseObject is byte)
                                        searchResults = tableToSearch.AsEnumerable().Where(i =>
                                            (byte)i[selectedIndex] <
                                            toSearchForByte);
                                    else if (baseObject is DateTime)
                                        searchResults = tableToSearch.AsEnumerable().Where(i =>
                                            (DateTime)i[selectedIndex] <
                                            toSearchForDateTime);
                                    else if (baseObject is decimal)
                                        searchResults = tableToSearch.AsEnumerable().Where(i =>
                                            (decimal)i[selectedIndex] <
                                            toSearchForDecimal);
                                    else
                                        throw new InvalidOperationException(
                                            "Comparator \"<\" cannot be applied to strings.");
                                    break;

                                case ">=":
                                    if (baseObject is byte)
                                        searchResults = tableToSearch.AsEnumerable().Where(i =>
                                            (byte)i[selectedIndex] >=
                                            toSearchForByte);
                                    else if (baseObject is DateTime)
                                        searchResults = tableToSearch.AsEnumerable().Where(i =>
                                            (DateTime)i[selectedIndex] >=
                                            toSearchForDateTime);
                                    else if (baseObject is decimal)
                                        searchResults = tableToSearch.AsEnumerable().Where(i =>
                                            (decimal)i[selectedIndex] >=
                                            toSearchForDecimal);
                                    else
                                        throw new InvalidOperationException(
                                            "Comparator \">=\" cannot be applied to strings.");
                                    break;

                                case "<=":
                                    if (baseObject is byte)
                                        searchResults = tableToSearch.AsEnumerable().Where(i =>
                                            (byte)i[selectedIndex] <=
                                            toSearchForByte);
                                    else if (baseObject is DateTime)
                                        searchResults = tableToSearch.AsEnumerable().Where(i =>
                                            (DateTime)i[selectedIndex] <=
                                            toSearchForDateTime);
                                    else if (baseObject is decimal)
                                        searchResults = tableToSearch.AsEnumerable().Where(i =>
                                            (decimal)i[selectedIndex] <=
                                            toSearchForDecimal);
                                    else
                                        throw new InvalidOperationException(
                                            "Comparator \"<=\" cannot be applied to strings.");
                                    break;

```

```
                                    default:
                                        throw new InvalidOperationException("Invalid comparator.");
                                }
                                resultCount = searchResults.Count();
                                gViewToDisplay.DataSource = searchResults.ToList();
                            }
                            break;

                        case Global.Enums.ItemTypes.PublicFacility:
                            viewToDisplay = 2;
                            gViewToDisplay = gViewPublicFacilityResults;
                            selectedIndex = ddlPublicFacility.SelectedIndex;
                            baseObject = database.PublicFacilities.First()[selectedIndex];
                            if (baseObject != null)
                            {
                                IEnumerable<PublicFacility> searchResults = null;
                                DbSet<PublicFacility> tableToSearch = database.PublicFacilities;
                                switch (comparator)
                                {
                                    case "|":
                                        searchResults = tableToSearch.AsEnumerable().Where(i =>
                                            i[selectedIndex].ToString().IndexOf(
                                            toSearchFor, StringComparison.OrdinalIgnoreCase) >= 0);
                                        break;

                                    case "!|":
                                        searchResults = tableToSearch.AsEnumerable().Where(i =>
                                            i[selectedIndex].ToString().IndexOf(
                                            toSearchFor, StringComparison.OrdinalIgnoreCase) < 0);
                                        break;

                                    case "==":
                                        if (baseObject is byte)
                                            searchResults = tableToSearch.AsEnumerable().Where(i =>
                                                (byte)i[selectedIndex] ==
                                                toSearchForByte);
                                        else if (baseObject is DateTime)
                                            searchResults = tableToSearch.AsEnumerable().Where(i =>
                                                (DateTime)i[selectedIndex] ==
                                                toSearchForDateTime);
                                        else if (baseObject is decimal)
                                            searchResults = tableToSearch.AsEnumerable().Where(i =>
                                                (decimal)i[selectedIndex] ==
                                                toSearchForDecimal);
                                        else
                                            searchResults = tableToSearch.AsEnumerable().Where(i =>
                                                ((string)i[selectedIndex]).ToLower() == toSearchFor.
                                                ToLower());
                                        break;

                                    case "!=":
                                        if (baseObject is byte)
                                            searchResults = tableToSearch.AsEnumerable().Where(i =>
                                                (byte)i[selectedIndex] !=
                                                toSearchForByte);
                                        else if (baseObject is DateTime)
                                            searchResults = tableToSearch.AsEnumerable().Where(i =>
                                                (DateTime)i[selectedIndex] !=
                                                toSearchForDateTime);
                                        else if (baseObject is decimal)
                                            searchResults = tableToSearch.AsEnumerable().Where(i =>
                                                (decimal)i[selectedIndex] !=
                                                toSearchForDecimal);
                                        else
                                            searchResults = tableToSearch.AsEnumerable().Where(i =>
                                                ((string)i[selectedIndex]).ToLower() != toSearchFor.
```

```
                                                ToLower());
                                        break;

                                    case ">":
                                        if (baseObject is byte)
                                            searchResults = tableToSearch.AsEnumerable().Where(i =>
                                                (byte)i[selectedIndex] >
                                                toSearchForByte);
                                        else if (baseObject is DateTime)
                                            searchResults = tableToSearch.AsEnumerable().Where(i =>
                                                (DateTime)i[selectedIndex] >
                                                toSearchForDateTime);
                                        else if (baseObject is decimal)
                                            searchResults = tableToSearch.AsEnumerable().Where(i =>
                                                (decimal)i[selectedIndex] >
                                                toSearchForDecimal);
                                        else
                                            throw new InvalidOperationException(
                                                "Comparator \">\" cannot be applied to strings.");
                                        break;

                                    case "<":
                                        if (baseObject is byte)
                                            searchResults = tableToSearch.AsEnumerable().Where(i =>
                                                (byte)i[selectedIndex] <
                                                toSearchForByte);
                                        else if (baseObject is DateTime)
                                            searchResults = tableToSearch.AsEnumerable().Where(i =>
                                                (DateTime)i[selectedIndex] <
                                                toSearchForDateTime);
                                        else if (baseObject is decimal)
                                            searchResults = tableToSearch.AsEnumerable().Where(i =>
                                                (decimal)i[selectedIndex] <
                                                toSearchForDecimal);
                                        else
                                            throw new InvalidOperationException(
                                                "Comparator \"<\" cannot be applied to strings.");
                                        break;

                                    case ">=":
                                        if (baseObject is byte)
                                            searchResults = tableToSearch.AsEnumerable().Where(i =>
                                                (byte)i[selectedIndex] >=
                                                toSearchForByte);
                                        else if (baseObject is DateTime)
                                            searchResults = tableToSearch.AsEnumerable().Where(i =>
                                                (DateTime)i[selectedIndex] >=
                                                toSearchForDateTime);
                                        else if (baseObject is decimal)
                                            searchResults = tableToSearch.AsEnumerable().Where(i =>
                                                (decimal)i[selectedIndex] >=
                                                toSearchForDecimal);
                                        else
                                            throw new InvalidOperationException(
                                                "Comparator \">=\" cannot be applied to strings.");
                                        break;

                                    case "<=":
                                        if (baseObject is byte)
                                            searchResults = tableToSearch.AsEnumerable().Where(i =>
                                                (byte)i[selectedIndex] <=
                                                toSearchForByte);
                                        else if (baseObject is DateTime)
                                            searchResults = tableToSearch.AsEnumerable().Where(i =>
                                                (DateTime)i[selectedIndex] <=
                                                toSearchForDateTime);
```

```
                                        else if (baseObject is decimal)
                                            searchResults = tableToSearch.AsEnumerable().Where(i =>
                                                (decimal)i[selectedIndex] <=
                                                toSearchForDecimal);
                                        else
                                            throw new InvalidOperationException(
                                                "Comparator \"<=\" cannot be applied to strings.");
                                        break;

                                    default:
                                        throw new InvalidOperationException("Invalid comparator.");
                                }
                                resultCount = searchResults.Count();
                                gViewToDisplay.DataSource = searchResults.ToList();
                            }
                            break;

                        default:
                            throw new InvalidOperationException(
                                "Invalid item type dropdown value.");
                    }
                    lblResult.Text = string.Format("{0} result{1} found.", resultCount,
                        resultCount == 1 ? "" : "s");
                    lblResult.Visible = true;
                    mViewSearchResults.ActiveViewIndex = viewToDisplay;
                    gViewToDisplay.DataBind();
                }
            }
            catch (Exception ex)
            {
                lblError.Text = "Error: " + ex.Message;
                lblError.Visible = true;
            }
        }

        /*-------------------------------------------------------------------------------------
         * Name:    ddlBusiness_SelectedIndexChanged
         * Type:    Event Handler Method
         * Purpose: Handles the change of visibility on controls when the user chooses a business
         *          field.
         * Input:   object sender, holds a reference to the object that raised this event.
         * Input:   EventArgs e, holds data related to this event.
         * Output:  Nothing.
        -------------------------------------------------------------------------------------*/
        protected void ddlBusiness_SelectedIndexChanged(object sender, EventArgs e)
        {
            // Hide things until needed.
            lblError.Visible = false;
            lblResult.Visible = false;

            ddlComparatorsStrings.Visible = false;
            ddlComparatorsStrings.SelectedIndex = 0;
            ddlComparatorsNotStrings.Visible = false;
            ddlComparatorsNotStrings.SelectedIndex = 0;

            txtSearch.Visible = false;
            txtSearch.Text = "";
            btnSearch.Visible = false;

            mViewSearchResults.ActiveViewIndex = -1;

            // Display the needed control.
            switch ((Global.Enums.BusinessFields)ddlBusiness.SelectedIndex)
            {
                case Global.Enums.BusinessFields.Name:
                case Global.Enums.BusinessFields.Type:
```

```
                case Global.Enums.BusinessFields.StreetAddress:
                case Global.Enums.BusinessFields.City:
                case Global.Enums.BusinessFields.State:
                case Global.Enums.BusinessFields.Zip:
                case Global.Enums.BusinessFields.Phone:
                case Global.Enums.BusinessFields.LicenseNumber:
                case Global.Enums.BusinessFields.LicenseStatus:
                case Global.Enums.BusinessFields.CouncilDistrict:
                    ddlComparatorsStrings.Visible = true;
                    break;

                case Global.Enums.BusinessFields.Latitude:
                case Global.Enums.BusinessFields.Longitude:
                case Global.Enums.BusinessFields.LicenseIssueDate:
                case Global.Enums.BusinessFields.LicenseExpirDate:
                    ddlComparatorsNotStrings.Visible = true;
                    break;

                default:
                    ddlComparatorsStrings.Visible = false;
                    ddlComparatorsNotStrings.Visible = false;
                    break;
            }
        }

        /*-------------------------------------------------------------------------------------
         * Name:    ddlComparatorsNotStrings_SelectedIndexChanged
         * Type:    Event Handler Method
         * Purpose: Handles the change of visibility on controls when the user chooses what type of
         *          comparison to make.
         * Input:   object sender, holds a reference to the object that raised this event.
         * Input:   EventArgs e, holds data related to this event.
         * Output:  Nothing.
        -------------------------------------------------------------------------------------*/
        protected void ddlComparatorsNotStrings_SelectedIndexChanged(object sender, EventArgs e)
        {
            // Hide things until needed.
            lblError.Visible = false;
            lblResult.Visible = false;

            txtSearch.Visible = false;
            txtSearch.Text = "";
            btnSearch.Visible = false;

            mViewSearchResults.ActiveViewIndex = -1;

            // Display the needed control.
            switch ((Global.Enums.ComparatorsNotStrings)ddlComparatorsNotStrings.SelectedIndex)
            {
                case Global.Enums.ComparatorsNotStrings.Contain:
                case Global.Enums.ComparatorsNotStrings.NotContain:
                case Global.Enums.ComparatorsNotStrings.Equal:
                case Global.Enums.ComparatorsNotStrings.NotEqual:
                case Global.Enums.ComparatorsNotStrings.Greater:
                case Global.Enums.ComparatorsNotStrings.Less:
                case Global.Enums.ComparatorsNotStrings.GreaterEqual:
                case Global.Enums.ComparatorsNotStrings.LessEqual:
                    txtSearch.Visible = true;
                    btnSearch.Visible = true;
                    break;

                default:
                    break;
            }
        }

```

```
        /*-------------------------------------------------------------------------------------
         * Name:    ddlComparatorsStrings_SelectedIndexChanged
         * Type:    Event Handler Method
         * Purpose: Handles the change of visibility on controls when the user chooses what type of
         *          comparison to make.
         * Input:   object sender, holds a reference to the object that raised this event.
         * Input:   EventArgs e, holds data related to this event.
         * Output:  Nothing.
        -------------------------------------------------------------------------------------*/
        protected void ddlComparatorsStrings_SelectedIndexChanged(object sender, EventArgs e)
        {
            // Hide things until needed.
            lblError.Visible = false;
            lblResult.Visible = false;

            txtSearch.Visible = false;
            txtSearch.Text = "";
            btnSearch.Visible = false;

            mViewSearchResults.ActiveViewIndex = -1;

            // Display the needed control.
            switch ((Global.Enums.ComparatorsStrings)ddlComparatorsStrings.SelectedIndex)
            {
                case Global.Enums.ComparatorsStrings.Contain:
                case Global.Enums.ComparatorsStrings.NotContain:
                case Global.Enums.ComparatorsStrings.Equal:
                case Global.Enums.ComparatorsStrings.NotEqual:
                    txtSearch.Visible = true;
                    btnSearch.Visible = true;
                    break;

                default:
                    break;
            }
        }

        /*-------------------------------------------------------------------------------------
         * Name:    ddlItemType_SelectedIndexChanged
         * Type:    Event Handler Method
         * Purpose: Handles showing and hiding the various controls to allow a user to search the
         *          database.
         * Input:   object sender, holds a reference to the object that raised this event.
         * Input:   EventArgs e, holds data related to this event.
         * Output:  Nothing.
        -------------------------------------------------------------------------------------*/
        protected void ddlItemType_SelectedIndexChanged(object sender, EventArgs e)
        {
            // Hide things until needed.
            lblError.Visible = false;
            lblResult.Visible = false;

            ddlBusiness.Visible = false;
            ddlBusiness.SelectedIndex = 0;
            ddlPark.Visible = false;
            ddlPark.SelectedIndex = 0;
            ddlPublicFacility.Visible = false;
            ddlPublicFacility.SelectedIndex = 0;
            ddlComparatorsStrings.Visible = false;
            ddlComparatorsStrings.SelectedIndex = 0;
            ddlComparatorsNotStrings.Visible = false;
            ddlComparatorsNotStrings.SelectedIndex = 0;

            txtSearch.Visible = false;
            txtSearch.Text = "";
            btnSearch.Visible = false;
```

```

            mViewSearchResults.ActiveViewIndex = -1;

            // Display the needed control.
            switch ((Global.Enums.ItemTypes)ddlItemType.SelectedIndex)
            {
                case Global.Enums.ItemTypes.Business:
                    ddlBusiness.Visible = true;
                    break;

                case Global.Enums.ItemTypes.Park:
                    ddlPark.Visible = true;
                    break;

                case Global.Enums.ItemTypes.PublicFacility:
                    ddlPublicFacility.Visible = true;
                    break;

                default:
                    break;
            }
        }

        /*-------------------------------------------------------------------------------------
         * Name:    ddlPark_SelectedIndexChanged
         * Type:    Event Handler Method
         * Purpose: Handles the change of visibility on controls when the user chooses a park field.
         * Input:   object sender, holds a reference to the object that raised this event.
         * Input:   EventArgs e, holds data related to this event.
         * Output:  Nothing.
        -------------------------------------------------------------------------------------*/
        protected void ddlPark_SelectedIndexChanged(object sender, EventArgs e)
        {
            // Hide things until needed.
            lblError.Visible = false;
            lblResult.Visible = false;

            ddlComparatorsStrings.Visible = false;
            ddlComparatorsStrings.SelectedIndex = 0;
            ddlComparatorsNotStrings.Visible = false;
            ddlComparatorsNotStrings.SelectedIndex = 0;

            txtSearch.Visible = false;
            txtSearch.Text = "";
            btnSearch.Visible = false;

            mViewSearchResults.ActiveViewIndex = -1;

            // Display the needed control.
            switch ((Global.Enums.ParkFields)ddlPark.SelectedIndex)
            {
                case Global.Enums.ParkFields.Name:
                case Global.Enums.ParkFields.Type:
                case Global.Enums.ParkFields.StreetAddress:
                case Global.Enums.ParkFields.City:
                case Global.Enums.ParkFields.State:
                case Global.Enums.ParkFields.Zip:
                case Global.Enums.ParkFields.Phone:
                    ddlComparatorsStrings.Visible = true;
                    break;

                case Global.Enums.ParkFields.Latitude:
                case Global.Enums.ParkFields.Longitude:
                case Global.Enums.ParkFields.FeatureBaseball:
                case Global.Enums.ParkFields.FeatureBasketball:
                case Global.Enums.ParkFields.FeatureGolf:
```

```
                case Global.Enums.ParkFields.FeatureLargeMPField:
                case Global.Enums.ParkFields.FeatureTennis:
                case Global.Enums.ParkFields.FeatureVolleyball:
                    ddlComparatorsNotStrings.Visible = true;
                    break;

                default:
                    break;
            }
        }

        /*-------------------------------------------------------------------------------------
         * Name:    ddlPublicFacility_SelectedIndexChanged
         * Type:    Event Handler Method
         * Purpose: Handles the change of visibility on controls when the user chooses a public
         *          facility field.
         * Input:   object sender, holds a reference to the object that raised this event.
         * Input:   EventArgs e, holds data related to this event.
         * Output:  Nothing.
        -------------------------------------------------------------------------------------*/
        protected void ddlPublicFacility_SelectedIndexChanged(object sender, EventArgs e)
        {
            // Hide things until needed.
            lblError.Visible = false;
            lblResult.Visible = false;

            ddlComparatorsStrings.Visible = false;
            ddlComparatorsStrings.SelectedIndex = 0;
            ddlComparatorsNotStrings.Visible = false;
            ddlComparatorsNotStrings.SelectedIndex = 0;

            txtSearch.Visible = false;
            txtSearch.Text = "";
            btnSearch.Visible = false;

            mViewSearchResults.ActiveViewIndex = -1;

            // Display the needed control.
            switch ((Global.Enums.PublicFacilityFields)ddlPublicFacility.SelectedIndex)
            {
                case Global.Enums.PublicFacilityFields.Name:
                case Global.Enums.PublicFacilityFields.Type:
                case Global.Enums.PublicFacilityFields.StreetAddress:
                case Global.Enums.PublicFacilityFields.City:
                case Global.Enums.PublicFacilityFields.State:
                case Global.Enums.PublicFacilityFields.Zip:
                case Global.Enums.PublicFacilityFields.Phone:
                    ddlComparatorsStrings.Visible = true;
                    break;

                case Global.Enums.PublicFacilityFields.Latitude:
                case Global.Enums.PublicFacilityFields.Longitude:
                    ddlComparatorsNotStrings.Visible = true;
                    break;

                default:
                    break;
            }
        }
    }
}
```

```
/*-----------------------------------------------------------------------------------------
 * Author:      Dan Cassidy
 * Date:        2015-06-23
 * Assignment:  cView-P4
 * Source File: Statistics.aspx.cs
 * Language:    C#
 * Course:      CSCI-C 490, C# Programming, MoWe 08:00
 * Purpose:     Code-behind file for Statistics.aspx.  Controls the statistics displayed.
-----------------------------------------------------------------------------------------*/

using System;
using System.Collections.Generic;
using System.Globalization;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace cView_P4_DanCassidy
{
    public partial class Statistics : System.Web.UI.Page
    {
        /*-----------------------------------------------------------------------------------------
         * Name:    Page_Load
         * Type:    Event Handler Method
         * Purpose: Handles anything that should happen on page load. In this case, it calculates
         *          and displays some statistics.
         * Input:   object sender, holds a reference to the object that raised this event.
         * Input:   EventArgs e, holds data related to this event.
         * Output:  Nothing.
        -----------------------------------------------------------------------------------------*/
        protected void Page_Load(object sender, EventArgs e)
        {
            try
            {
                using (CViewDataEntities database = new CViewDataEntities())
                {
                    // Hijacked from http://stackoverflow.com/a/1206029.
                    TextInfo textInfo = new CultureInfo("en-US", false).TextInfo;

                    // Total number of parks.
                    lblStatistics1.Text = database.Parks.Count().ToString();

                    // Total number of parks, grouped by park type.
                    lblStatistics2.Text = "";
                    var parksByType = database.Parks.GroupBy(p => p.Type);
                    foreach (var parkType in parksByType)
                        lblStatistics2.Text += (parkType.Key == "" ? "(Empty)" :
                            textInfo.ToTitleCase(parkType.Key.ToLower())) + ": " +
                            parkType.Count() + "<br />";

                    // Total number of businesses.
                    lblStatistics3.Text = database.Businesses.Count().ToString();

                    // Total number of license renewals for each business.
                    lblStatistics4.Text = "";
                    var businessRenewals = database.Businesses.Where(
                        b => b.LicenseStatus == "Renewed").GroupBy(b => b.Name);
                    foreach (var business in businessRenewals)
                        lblStatistics4.Text += (business.Key == "" ? "(Empty)" :
                            textInfo.ToTitleCase(business.Key.ToLower())) + ": " +
                            business.Count() + "<br />";

                    // Total number of facilities that have the substring "Fire"
                    lblStatistics5.Text = database.PublicFacilities.Where(pf => pf.Name.Contains(
                        "fire")).Count().ToString();
```

```
                }
            }
            catch
            {
                lblError.Text = "Error: Could not calculate statistics.";
                lblError.Visible = true;
            }
        }
    }
}
```

```
/*-----------------------------------------------------------------------------------------------
 * Author:      Dan Cassidy
 * Date:        2015-06-23
 * Assignment:  cView-P4
 * Source File: CViewDataModelExtension.cs
 * Language:    C#
 * Course:      CSCI-C 490, C# Programming, MoWe 08:00
 * Purpose:     Extends the Business, Park, and PublicFacility class generated by the ASP.NET Entity
 *              Framework for this solution.
-----------------------------------------------------------------------------------------------*/

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace cView_P4_DanCassidy
{
    public partial class Business
    {
        /*-----------------------------------------------------------------------------------------
         * Name:    this[]
         * Type:    Indexer
         * Purpose: Provides easy access to the properties of the class.
         * Input:   int fiendNum, represents the desired property.
         * Output:  object, contains whichever property was desired, or null if the property was not
         *          found.
        -----------------------------------------------------------------------------------------*/
        public object this[int fieldNum]
        {
            get
            {
                switch ((Global.Enums.BusinessFields)fieldNum)
                {
                    case Global.Enums.BusinessFields.Name:
                        return Name;
                    case Global.Enums.BusinessFields.Type:
                        return Type;
                    case Global.Enums.BusinessFields.StreetAddress:
                        return StreetAddress;
                    case Global.Enums.BusinessFields.City:
                        return City;
                    case Global.Enums.BusinessFields.State:
                        return State;
                    case Global.Enums.BusinessFields.Zip:
                        return Zip;
                    case Global.Enums.BusinessFields.Latitude:
                        return Latitude;
                    case Global.Enums.BusinessFields.Longitude:
                        return Longitude;
                    case Global.Enums.BusinessFields.Phone:
                        return Phone;
                    case Global.Enums.BusinessFields.LicenseNumber:
                        return LicenseNumber;
                    case Global.Enums.BusinessFields.LicenseIssueDate:
                        return LicenseIssueDate;
                    case Global.Enums.BusinessFields.LicenseExpirDate:
                        return LicenseExpirDate;
                    case Global.Enums.BusinessFields.LicenseStatus:
                        return LicenseStatus;
                    case Global.Enums.BusinessFields.CouncilDistrict:
                        return CouncilDistrict;
                    default:
                        return null;
                }
            }
```

```
        }
    }

    public partial class Park
    {
        /*-------------------------------------------------------------------------------------
         * Name:    this[]
         * Type:    Indexer
         * Purpose: Provides easy access to the properties of the class.
         * Input:   int fiendNum, represents the desired property.
         * Output:  object, contains whichever property was desired, or null if the property was not
         *          found.
        -------------------------------------------------------------------------------------*/
        public object this[int fieldNum]
        {
            get
            {
                switch ((Global.Enums.ParkFields)fieldNum)
                {
                    case Global.Enums.ParkFields.Name:
                        return Name;
                    case Global.Enums.ParkFields.Type:
                        return Type;
                    case Global.Enums.ParkFields.StreetAddress:
                        return StreetAddress;
                    case Global.Enums.ParkFields.City:
                        return City;
                    case Global.Enums.ParkFields.State:
                        return State;
                    case Global.Enums.ParkFields.Zip:
                        return Zip;
                    case Global.Enums.ParkFields.Latitude:
                        return Latitude;
                    case Global.Enums.ParkFields.Longitude:
                        return Longitude;
                    case Global.Enums.ParkFields.Phone:
                        return Phone;
                    case Global.Enums.ParkFields.FeatureBaseball:
                        return FeatureBaseball;
                    case Global.Enums.ParkFields.FeatureBasketball:
                        return FeatureBasketball;
                    case Global.Enums.ParkFields.FeatureGolf:
                        return FeatureGolf;
                    case Global.Enums.ParkFields.FeatureLargeMPField:
                        return FeatureLargeMPField;
                    case Global.Enums.ParkFields.FeatureTennis:
                        return FeatureTennis;
                    case Global.Enums.ParkFields.FeatureVolleyball:
                        return FeatureVolleyball;
                    default:
                        return null;
                }
            }
        }
    }

    public partial class PublicFacility
    {
        /*-------------------------------------------------------------------------------------
         * Name:    this[]
         * Type:    Indexer
         * Purpose: Provides easy access to the properties of the class.
         * Input:   int fiendNum, represents the desired property.
         * Output:  object, contains whichever property was desired, or null if the property was not
         *          found.
        -------------------------------------------------------------------------------------*/
```

```
        public object this[int fieldNum]
        {
            get
            {
                switch ((Global.Enums.PublicFacilityFields)fieldNum)
                {
                    case Global.Enums.PublicFacilityFields.Name:
                        return Name;
                    case Global.Enums.PublicFacilityFields.Type:
                        return Type;
                    case Global.Enums.PublicFacilityFields.StreetAddress:
                        return StreetAddress;
                    case Global.Enums.PublicFacilityFields.City:
                        return City;
                    case Global.Enums.PublicFacilityFields.State:
                        return State;
                    case Global.Enums.PublicFacilityFields.Zip:
                        return Zip;
                    case Global.Enums.PublicFacilityFields.Latitude:
                        return Latitude;
                    case Global.Enums.PublicFacilityFields.Longitude:
                        return Longitude;
                    case Global.Enums.PublicFacilityFields.Phone:
                        return Phone;
                    default:
                        return null;
                }
            }
        }
    }
}
```

```
/*-----------------------------------------------------------------------------------------
 * Author:      Dan Cassidy
 * Date:        2015-06-23
 * Assignment:  cView-P4
 * Source File: SimpleConvert.cs
 * Language:    C#
 * Course:      CSCI-C 490, C# Programming, MoWe 08:00
 * Purpose:     Provides simplified variants and extensions of some Convert methods.
-----------------------------------------------------------------------------------------*/

namespace System
{
    public static class SimpleConvert
    {
        /*-----------------------------------------------------------------------------------
         * Name:    ToByte
         * Type:    Method
         * Purpose: Attempts to convert the given parameter, but returns the default object value if
         *          it fails for any reason.
         * Input:   string value, containing the value on which conversion will be attempted.
         * Output:  byte object containing either the converted value or the default object value.
        -----------------------------------------------------------------------------------*/
        public static byte ToByte(string value)
        {
            try
            {
                return Convert.ToByte(value);
            }
            catch
            {
                return default(byte);
            }
        }

        /*-----------------------------------------------------------------------------------
         * Name:    ToByteN
         * Type:    Method
         * Purpose: Attempts to convert the given parameter, but returns a null if it fails for any
         *          reason.
         * Input:   string value, containing the value on which conversion will be attempted.
         * Output:  Nullable<byte> object containing either the converted value or a null.
        -----------------------------------------------------------------------------------*/
        public static Nullable<byte> ToByteN(string value)
        {
            try
            {
                return Convert.ToByte(value);
            }
            catch
            {
                return null;
            }
        }

        /*-----------------------------------------------------------------------------------
         * Name:    ToDateTime
         * Type:    Method
         * Purpose: Attempts to convert the given parameter, but returns the default object value if
         *          it fails for any reason.
         * Input:   string value, containing the value on which conversion will be attempted.
         * Output:  DateTime object containing either the converted value or the default object
         *          value.
        -----------------------------------------------------------------------------------*/
        public static DateTime ToDateTime(string value)
        {
            try
```

```
            {
                return Convert.ToDateTime(value);
            }
            catch
            {
                return default(DateTime);
            }
        }

        /*-------------------------------------------------------------------------------------
         * Name:    ToDateTimeN
         * Type:    Method
         * Purpose: Attempts to convert the given parameter, but returns a null if it fails for any
         *          reason.
         * Input:   string value, containing the value on which conversion will be attempted.
         * Output:  Nullable<DateTime> object containing either the converted value or a null.
        -------------------------------------------------------------------------------------*/
        public static Nullable<DateTime> ToDateTimeN(string value)
        {
            try
            {
                return Convert.ToDateTime(value);
            }
            catch
            {
                return null;
            }
        }

        /*-------------------------------------------------------------------------------------
         * Name:    ToDecimal
         * Type:    Method
         * Purpose: Attempts to convert the given parameter, but returns the default object value if
         *          it fails for any reason.
         * Input:   string value, containing the value on which conversion will be attempted.
         * Output:  Decimal object containing either the converted value or the default object
         *          value.
        -------------------------------------------------------------------------------------*/
        public static decimal ToDecimal(string value)
        {
            try
            {
                return Convert.ToDecimal(value);
            }
            catch
            {
                return default(decimal);
            }
        }

        /*-------------------------------------------------------------------------------------
         * Name:    ToDecimalN
         * Type:    Method
         * Purpose: Attempts to convert the given parameter, but returns a null if it fails for any
         *          reason.
         * Input:   string value, containing the value on which conversion will be attempted.
         * Output:  Nullable<Decimal> object containing either the converted value or a null.
        -------------------------------------------------------------------------------------*/
        public static Nullable<decimal> ToDecimalN(string value)
        {
            try
            {
                return Convert.ToDecimal(value);
            }
            catch
            {
```

```
                return null;
            }
        }

        /*-----------------------------------------------------------------------------------
         * Name:    ToInt32
         * Type:    Method
         * Purpose: Attempts to convert the given parameter, but returns the default object value if
         *          it fails for any reason.
         * Input:   string value, containing the value on which conversion will be attempted.
         * Output:  int object containing either the converted value or the default object value.
        -----------------------------------------------------------------------------------*/
        public static int ToInt32(string value)
        {
            try
            {
                return Convert.ToInt32(value);
            }
            catch
            {
                return default(int);
            }
        }

        /*-----------------------------------------------------------------------------------
         * Name:    ToInt32N
         * Type:    Method
         * Purpose: Attempts to convert the given parameter, but returns a null if it fails for any
         *          reason.
         * Input:   string value, containing the value on which conversion will be attempted.
         * Output:  Nullable<int> object containing either the converted value or a null.
        -----------------------------------------------------------------------------------*/
        public static Nullable<int> ToInt32N(string value)
        {
            try
            {
                return Convert.ToInt32(value);
            }
            catch
            {
                return null;
            }
        }

        /*-----------------------------------------------------------------------------------
         * Name:    ToSingle
         * Type:    Method
         * Purpose: Attempts to convert the given parameter, but returns the default object value if
         *          it fails for any reason.
         * Input:   string value, containing the value on which conversion will be attempted.
         * Output:  float object containing either the converted value or the default object value.
        -----------------------------------------------------------------------------------*/
        public static float ToSingle(string value)
        {
            try
            {
                return Convert.ToSingle(value);
            }
            catch
            {
                return default(float);
            }
        }

        /*-----------------------------------------------------------------------------------
         * Name:    ToSingleN
```

```
         * Type:    Method
         * Purpose: Attempts to convert the given parameter, but returns the default object value if
         *          it fails for any reason.
         * Input:   string value, containing the value on which conversion will be attempted.
         * Output:  Nullable<float> object containing either the converted value or a null.
        ----------------------------------------------------------------------------------------*/
        public static Nullable<float> ToSingleN(string value)
        {
            try
            {
                return Convert.ToSingle(value);
            }
            catch
            {
                return null;
            }
        }
    }
}

```

```
/*-----------------------------------------------------------------------------------------
 * Author:      Dan Cassidy
 * Date:        2015-06-23
 * Assignment:  cView-P4
 * Source File: Global.asax.cs
 * Language:    C#
 * Course:      CSCI-C 490, C# Programming, MoWe 08:00
 * Purpose:     Code-behind file for Global.asax.  Contains things that should be globally
 *              accessible, such as enums, exceptions, and strings.
-----------------------------------------------------------------------------------------*/

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Security;
using System.Web.SessionState;

namespace cView_P4_DanCassidy
{
    public class Global : System.Web.HttpApplication
    {
        public static class Enums
        {
            /*-----------------------------------------------------------------------------------
             * Name:    BusinessFields
             * Type:    Enum
             * Purpose: Represents the different fields present in the Business class.
            -----------------------------------------------------------------------------------*/
            public enum BusinessFields
            {
                Name = 1,
                Type,
                StreetAddress,
                City,
                State,
                Zip,
                Latitude,
                Longitude,
                Phone,
                LicenseNumber,
                LicenseIssueDate,
                LicenseExpirDate,
                LicenseStatus,
                CouncilDistrict
            }

            /*-----------------------------------------------------------------------------------
             * Name:    ComparatorsNotStrings
             * Type:    Enum
             * Purpose: Represents the possible comparators available for use on non-strings.
            -----------------------------------------------------------------------------------*/
            public enum ComparatorsNotStrings
            {
                Contain = 1,
                NotContain,
                Equal,
                NotEqual,
                Greater,
                Less,
                GreaterEqual,
                LessEqual
            }

            /*-----------------------------------------------------------------------------------
             * Name:    ComparatorsStrings
```

```
             * Type:    Enum
             * Purpose: Represents the possible comparators available for use on strings.
            ----------------------------------------------------------------------------------*/
            public enum ComparatorsStrings
            {
                Contain = 1,
                NotContain,
                Equal,
                NotEqual
            }

            /*----------------------------------------------------------------------------------
             * Name:    ItemTypes
             * Type:    Enum
             * Purpose: Represents the different types of items.
            ----------------------------------------------------------------------------------*/
            public enum ItemTypes
            {
                Business = 1,
                Park,
                PublicFacility,
            }

            /*----------------------------------------------------------------------------------
             * Name:    ParkFields
             * Type:    Enum
             * Purpose: Represents the different fields present in the Park class.
            ----------------------------------------------------------------------------------*/
            public enum ParkFields
            {
                Name = 1,
                Type,
                StreetAddress,
                City,
                State,
                Zip,
                Latitude,
                Longitude,
                Phone,
                FeatureBaseball,
                FeatureBasketball,
                FeatureGolf,
                FeatureLargeMPField,
                FeatureTennis,
                FeatureVolleyball
            }

            /*----------------------------------------------------------------------------------
             * Name:    PublicFacilityFields
             * Type:    Enum
             * Purpose: Represents the different fiels present in the Public Facility class.
            ----------------------------------------------------------------------------------*/
            public enum PublicFacilityFields
            {
                Name = 1,
                Type,
                StreetAddress,
                City,
                State,
                Zip,
                Latitude,
                Longitude,
                Phone
            }

        }
```

```

        public class Exceptions
        {
            /*-------------------------------------------------------------------------------
             * Name:    DuplicatePKException
             * Type:    Exception
             * Purpose: Intended to describe a situation where an object with a duplicate primary
             *          key attempted to be inserted into a primary keyed data structure.
            -------------------------------------------------------------------------------*/
            [Serializable]
            public class DuplicatePKException : Exception
            {
                public DuplicatePKException() { }

                public DuplicatePKException(string message)
                    : base(message) { }

                public DuplicatePKException(string keyName, object keyValue)
                    : base(string.Format("An item already exists with a {0} of {1}.",
                           keyName, keyValue)) { }

                public DuplicatePKException(string message, Exception inner)
                    : base(message, inner) { }

                protected DuplicatePKException(
                  System.Runtime.Serialization.SerializationInfo info,
                  System.Runtime.Serialization.StreamingContext context)
                    : base(info, context) { }
            }

            /*-------------------------------------------------------------------------------
             * Name:    EmptyOrNullPKException
             * Type:    Exception
             * Purpose: Inteded to describe a situation where an object with an empty or null
             *          primary key attempted to be inserted into a primary keyed data structure.
            -------------------------------------------------------------------------------*/
            [Serializable]
            public class EmptyOrNullPKException : Exception
            {
                public EmptyOrNullPKException() { }

                public EmptyOrNullPKException(string keyName)
                    : base(string.Format("{0} cannot be empty or null.", keyName)) { }

                public EmptyOrNullPKException(string message, Exception inner)
                    : base(message, inner) { }

                protected EmptyOrNullPKException(
                  System.Runtime.Serialization.SerializationInfo info,
                  System.Runtime.Serialization.StreamingContext context)
                    : base(info, context) { }
            }
        }

        /*-------------------------------------------------------------------------------
         * Name:    Strings
         * Type:    Class
         * Purpose: Contains common strings used throughout the application in a centrally-managed
         *          location.
        -------------------------------------------------------------------------------*/
        public static class Strings
        {
            public const string Separator = ":";

            public const string BusinessName = "Business Name";
            public const string ParkName = "Park Name";
```

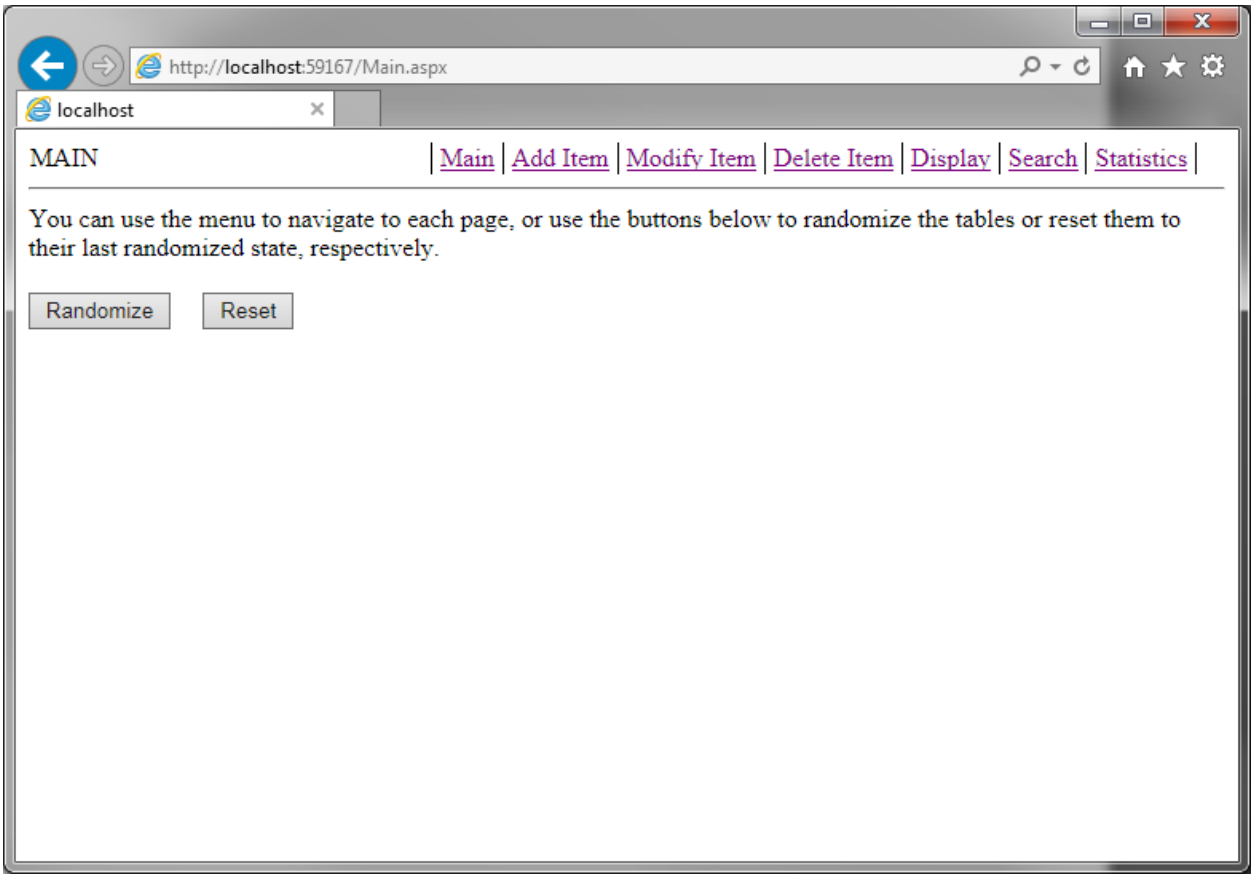
```
            public const string PublicFacilityName = "Public Facility Name";

            public const string BusinessType = "Type of Business";
            public const string ParkType = "Type of Park";
            public const string PublicFacilityType = "Type of Public Facility";

            public const string BusinessKey = "License Number";
            public const string ParkKey = ParkName;
            public const string PublicFacilityKey = PublicFacilityName;
        }

    }
}
```

```
-- NOTE:    The other two tables I used that dealt with business objects,
--          BusinessBase and BusinessReset are created in the exact same
--          manner (allowing for differing table names and constraint names,
--          so I am omitting their SQL statements for the sake of brevity.
USE [djcassid]
GO

SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

SET ANSI_PADDING ON
GO

CREATE TABLE [dbo].[Business](
    [Name] [varchar](100) NOT NULL,
    [Type] [varchar](50) NOT NULL,
    [StreetAddress] [varchar](50) NOT NULL,
    [City] [varchar](50) NOT NULL,
    [State] [varchar](50) NOT NULL,
    [Zip] [varchar](50) NOT NULL,
    [Latitude] [decimal](18, 15) NOT NULL,
    [Longitude] [decimal](18, 15) NOT NULL,
    [Phone] [varchar](50) NOT NULL,
    [LicenseNumber] [varchar](50) NOT NULL,
    [LicenseIssueDate] [date] NOT NULL,
    [LicenseExpirDate] [date] NOT NULL,
    [LicenseStatus] [varchar](50) NOT NULL,
    [CouncilDistrict] [varchar](50) NOT NULL,
 CONSTRAINT [PK_Business_LicenseNumber] PRIMARY KEY CLUSTERED
(
    [LicenseNumber] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]

GO

SET ANSI_PADDING OFF
GO

ALTER TABLE [dbo].[Business] ADD  CONSTRAINT [DF_Business_Latitude]  DEFAULT ((0)) FOR [Latitude]
GO

ALTER TABLE [dbo].[Business] ADD  CONSTRAINT [DF_Business_Longitude]  DEFAULT ((0)) FOR [Longitude]
GO

ALTER TABLE [dbo].[Business] ADD  CONSTRAINT [DF_Business_LicenseIssueDate]  DEFAULT ('0001-01-01') FOR [LicenseIssueDate]
GO

ALTER TABLE [dbo].[Business] ADD  CONSTRAINT [DF_Business_LicenseExpirDate]  DEFAULT ('0001-01-01') FOR [LicenseExpirDate]
GO



-- NOTE:    The other two tables I used that dealt with park objects,
--          ParkBase and ParkReset are created in the exact same
--          manner (allowing for differing table names and constraint names,
--          so I am omitting their SQL statements for the sake of brevity.
USE [djcassid]
```

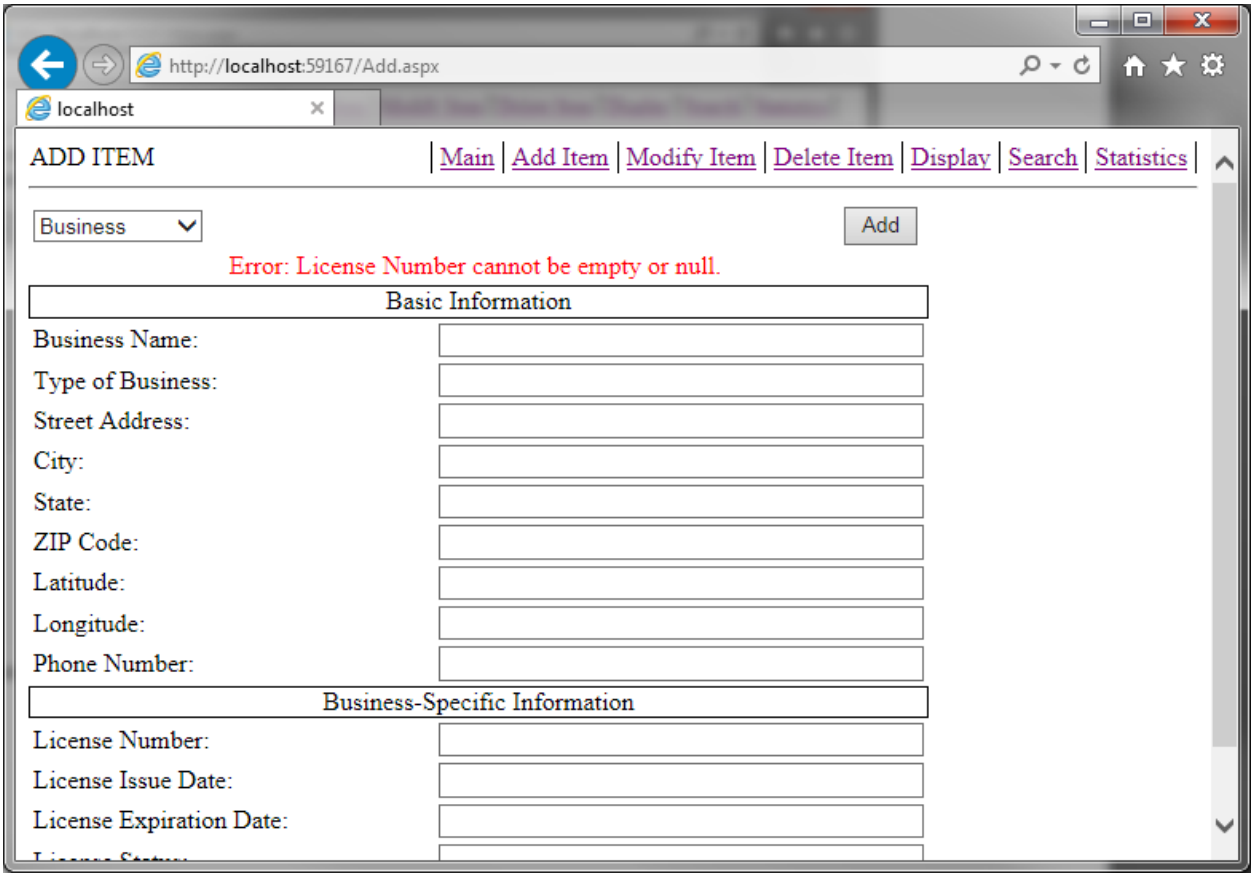
```
GO

SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

SET ANSI_PADDING ON
GO

CREATE TABLE [dbo].[Park](
    [Name] [varchar](100) NOT NULL,
    [Type] [varchar](50) NOT NULL,
    [StreetAddress] [varchar](50) NOT NULL,
    [City] [varchar](50) NOT NULL,
    [State] [varchar](50) NOT NULL,
    [Zip] [varchar](50) NOT NULL,
    [Latitude] [decimal](18, 15) NOT NULL,
    [Longitude] [decimal](18, 15) NOT NULL,
    [Phone] [varchar](50) NOT NULL,
    [FeatureBaseball] [tinyint] NOT NULL,
    [FeatureBasketball] [decimal](4, 1) NOT NULL,
    [FeatureGolf] [decimal](4, 1) NOT NULL,
    [FeatureLargeMPField] [tinyint] NOT NULL,
    [FeatureTennis] [tinyint] NOT NULL,
    [FeatureVolleyball] [tinyint] NOT NULL,
 CONSTRAINT [PK_Park_Name] PRIMARY KEY CLUSTERED
(
    [Name] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]

GO

SET ANSI_PADDING OFF
GO
ALTER TABLE [dbo].[Park] ADD  CONSTRAINT [DF_Park_Latitude]  DEFAULT ((0)) FOR [Latitude]
GO

ALTER TABLE [dbo].[Park] ADD  CONSTRAINT [DF_Park_Longitude]  DEFAULT ((0)) FOR [Longitude]
GO

ALTER TABLE [dbo].[Park] ADD  CONSTRAINT [DF_Park_FeatureBaseball]  DEFAULT ((0)) FOR [FeatureBaseball]
GO

ALTER TABLE [dbo].[Park] ADD  CONSTRAINT [DF_Park_FeatureBasketball]  DEFAULT ((0)) FOR [FeatureBasketball]
GO

ALTER TABLE [dbo].[Park] ADD  CONSTRAINT [DF_Park_FeatureGolf]  DEFAULT ((0)) FOR [FeatureGolf]
GO

ALTER TABLE [dbo].[Park] ADD  CONSTRAINT [DF_Park_FeatureLargeMPField]  DEFAULT ((0)) FOR [FeatureLargeMPField]
GO

ALTER TABLE [dbo].[Park] ADD  CONSTRAINT [DF_Park_FeatureTennis]  DEFAULT ((0)) FOR [FeatureTennis]
GO

ALTER TABLE [dbo].[Park] ADD  CONSTRAINT [DF_Park_FeatureVolleyball]  DEFAULT ((0)) FOR [FeatureVolleyball]
GO
```

```
-- NOTE:    The other two tables I used that dealt with public facility objects,
--          PublicFacilityBase and PublicFacilityReset are created in the exact same
--          manner (allowing for differing table names and constraint names,
--          so I am omitting their SQL statements for the sake of brevity.
USE [djcassid]
GO

SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

SET ANSI_PADDING ON
GO

CREATE TABLE [dbo].[PublicFacility](
    [Name] [varchar](100) NOT NULL,
    [Type] [varchar](50) NOT NULL,
    [StreetAddress] [varchar](50) NOT NULL,
    [City] [varchar](50) NOT NULL,
    [State] [varchar](50) NOT NULL,
    [Zip] [varchar](50) NOT NULL,
    [Latitude] [decimal](18, 15) NOT NULL,
    [Longitude] [decimal](18, 15) NOT NULL,
    [Phone] [varchar](50) NOT NULL,
 CONSTRAINT [PK_PublicFacility_Name] PRIMARY KEY CLUSTERED
(
    [Name] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]

GO

SET ANSI_PADDING OFF
GO

ALTER TABLE [dbo].[PublicFacility] ADD  CONSTRAINT [DF_PublicFacility_Latitude]  DEFAULT ((0)) FOR [Latitude]
GO

ALTER TABLE [dbo].[PublicFacility] ADD  CONSTRAINT [DF_PublicFacility_Longitude]  DEFAULT ((0)) FOR [Longitude]
GO
```

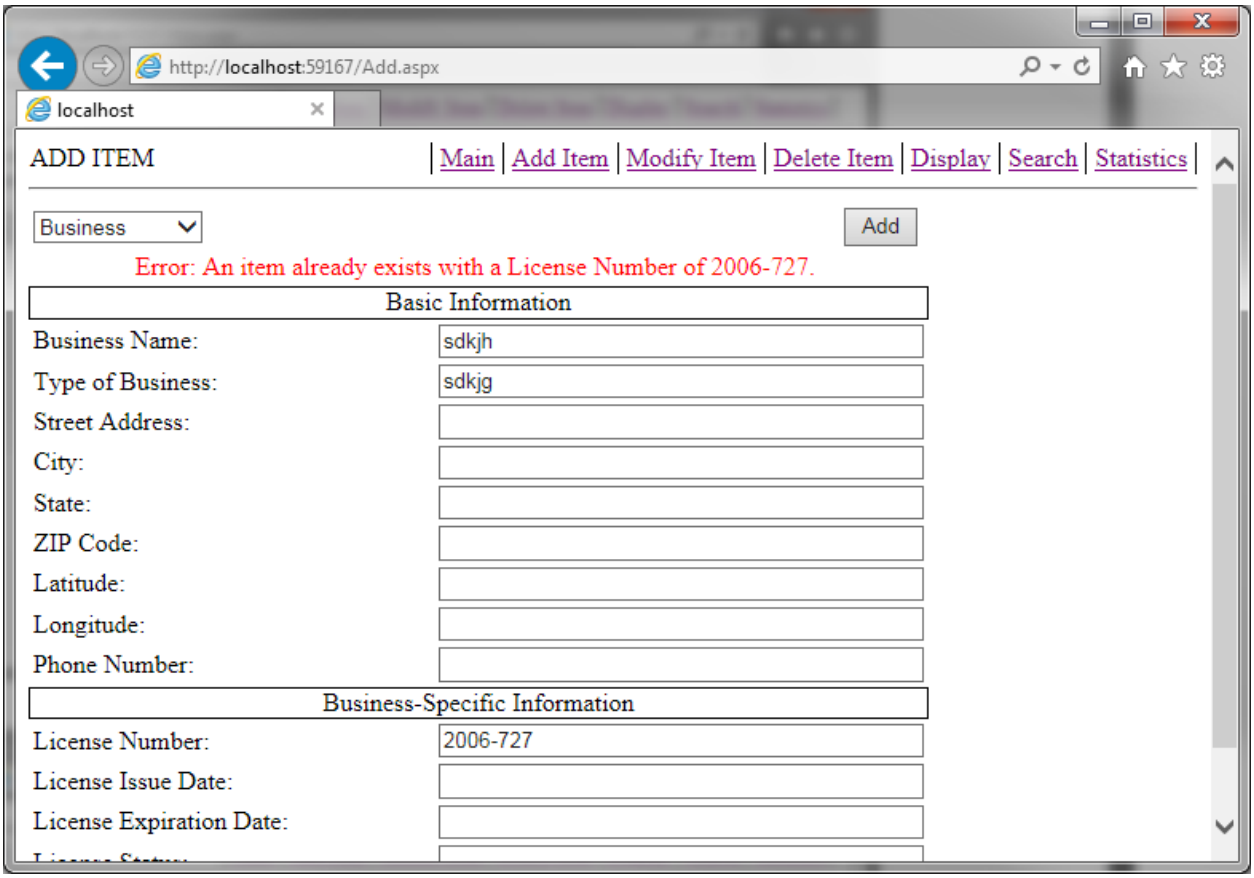
Main/Menu page.



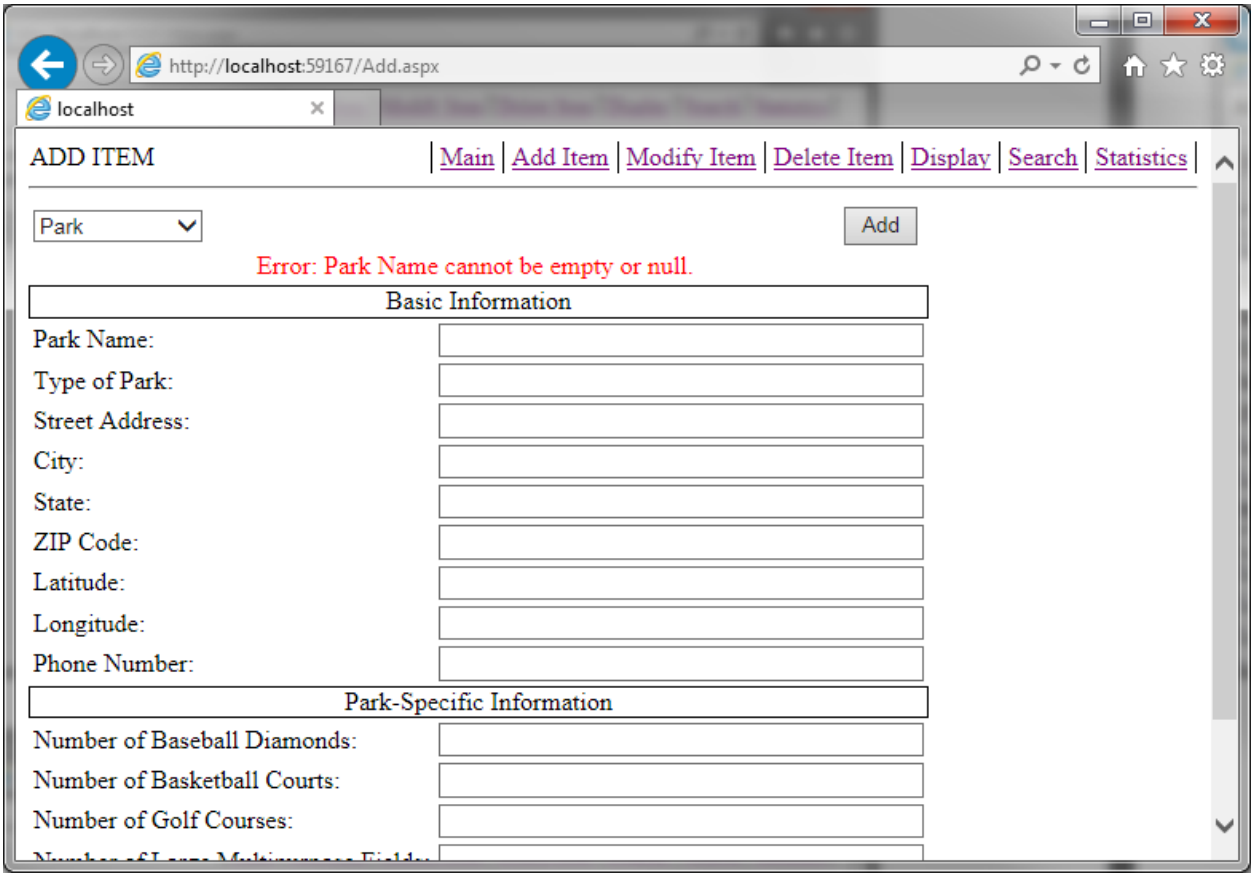
Attempting to add a business without a license number



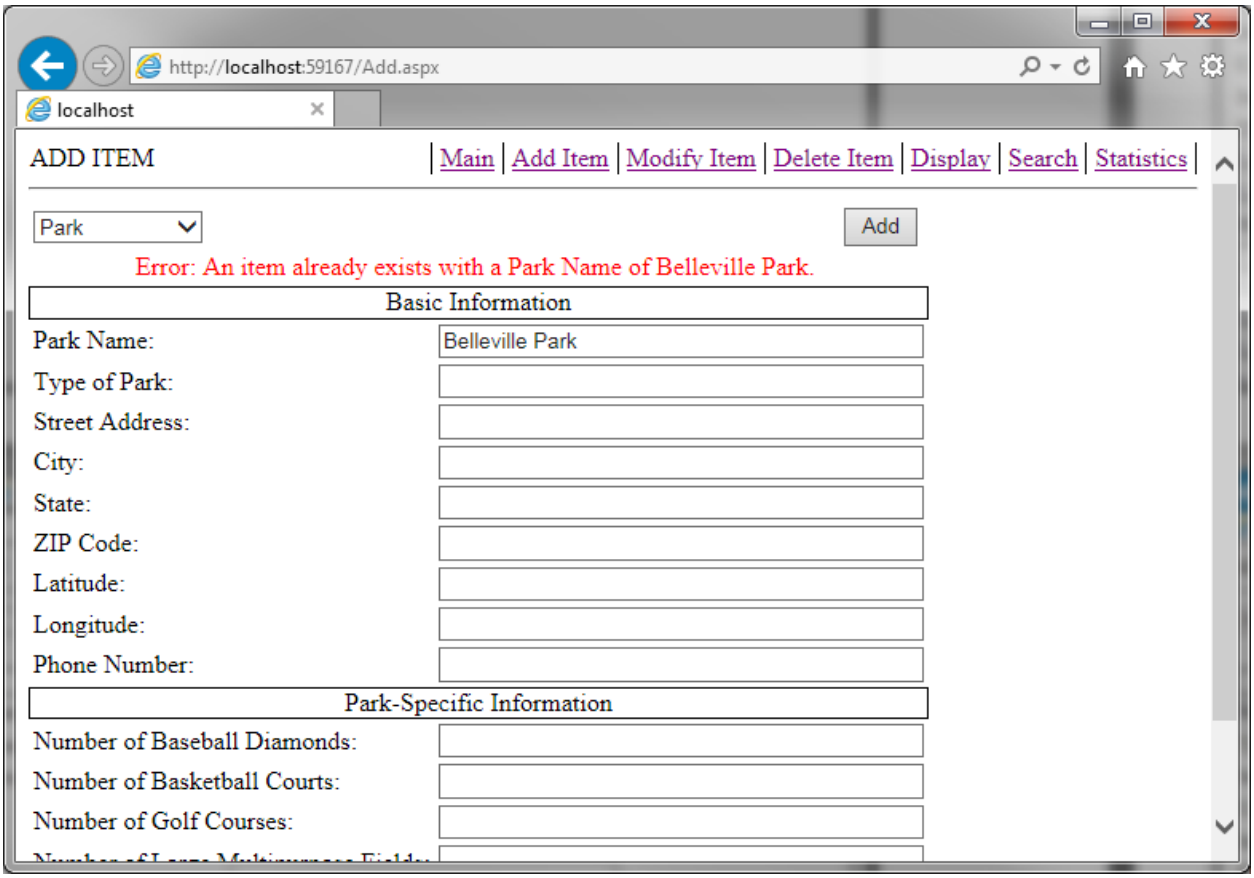
Attempting to add a business with a license number that already exists



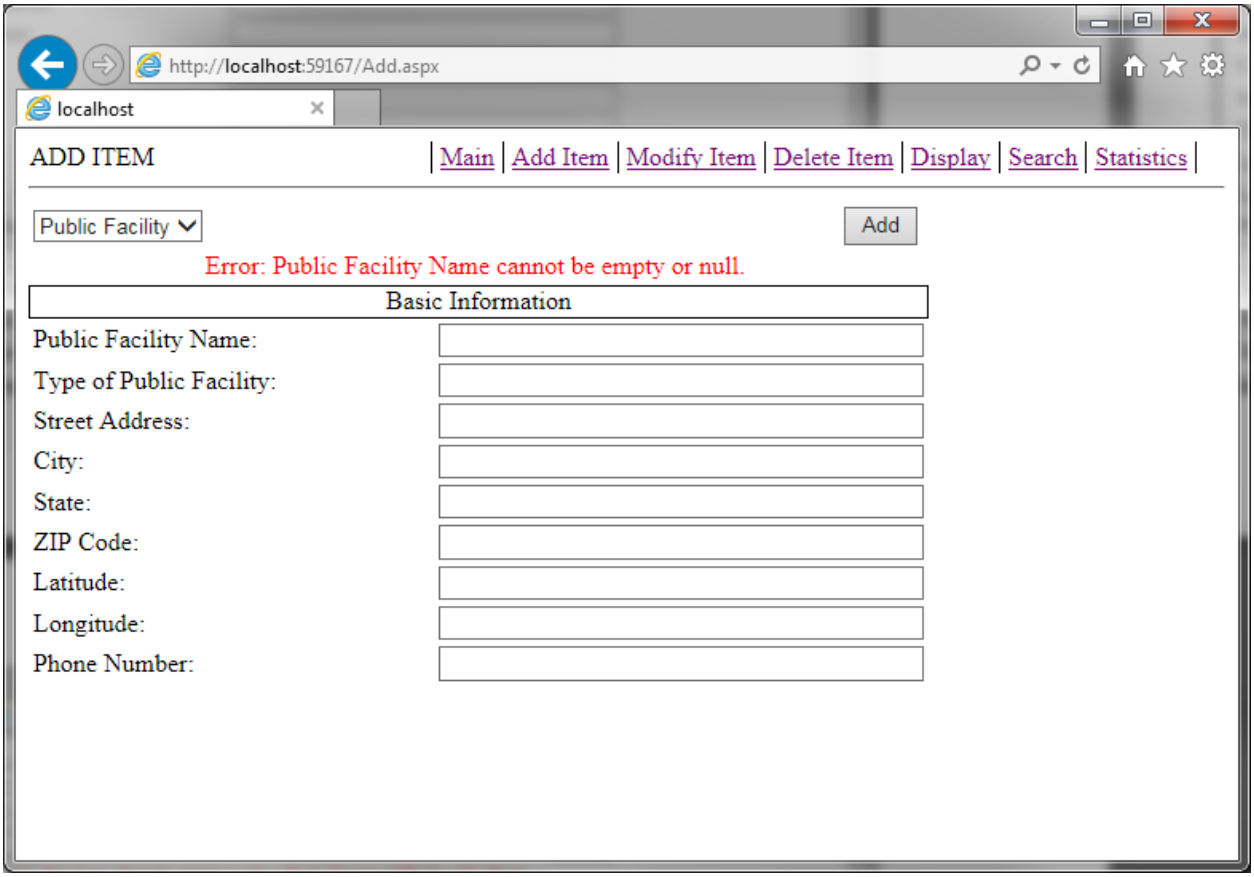
Attempting to add a park without a name



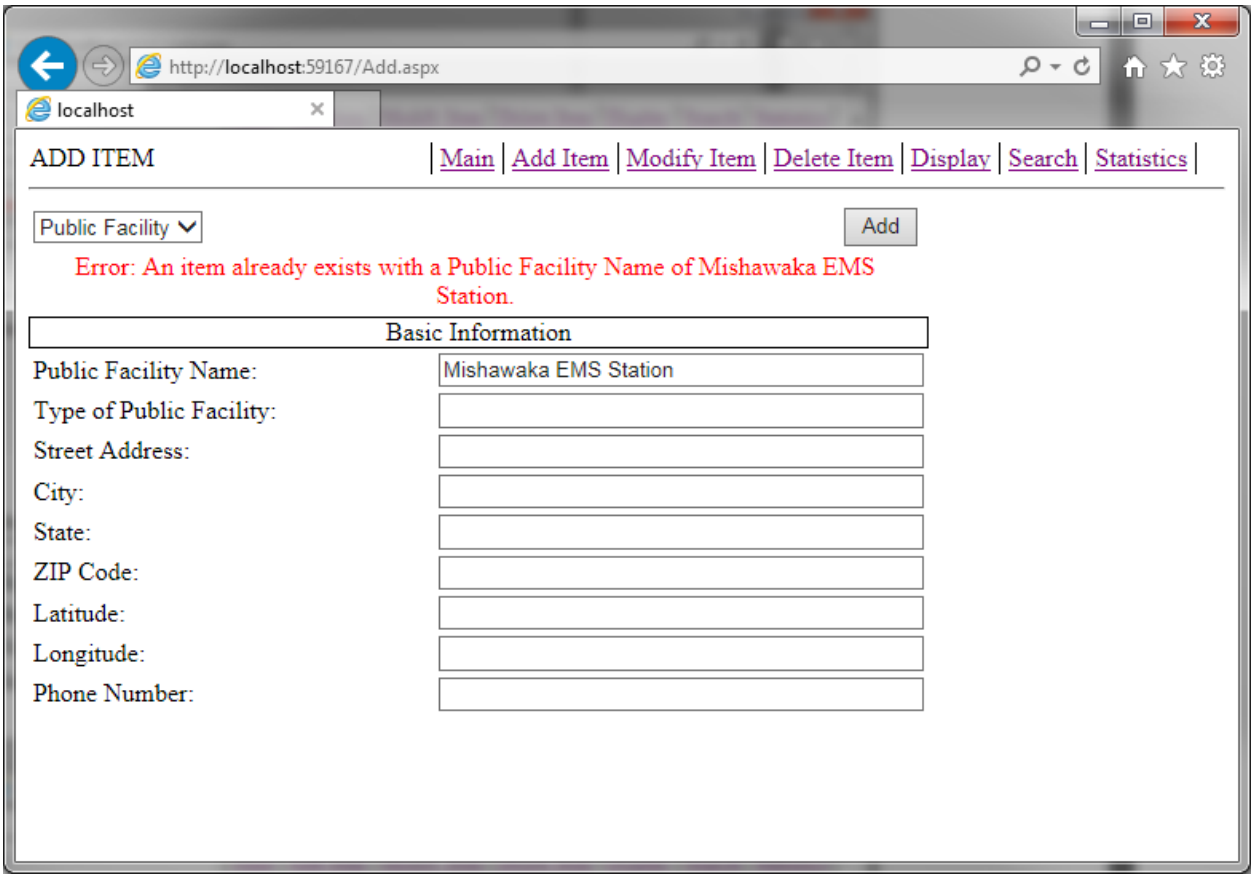
Attempting to add a park with name that already exists



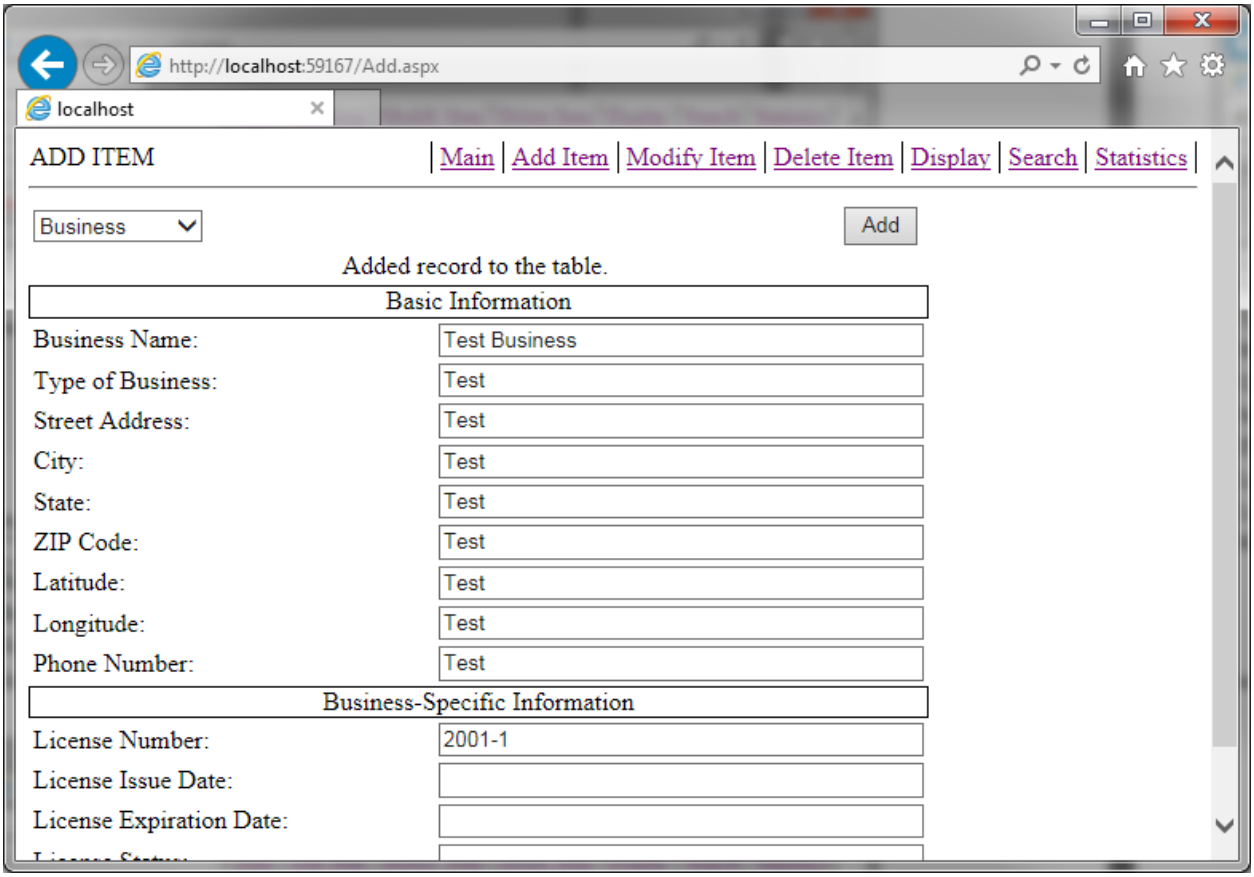
Attempting to add a public facility without a name



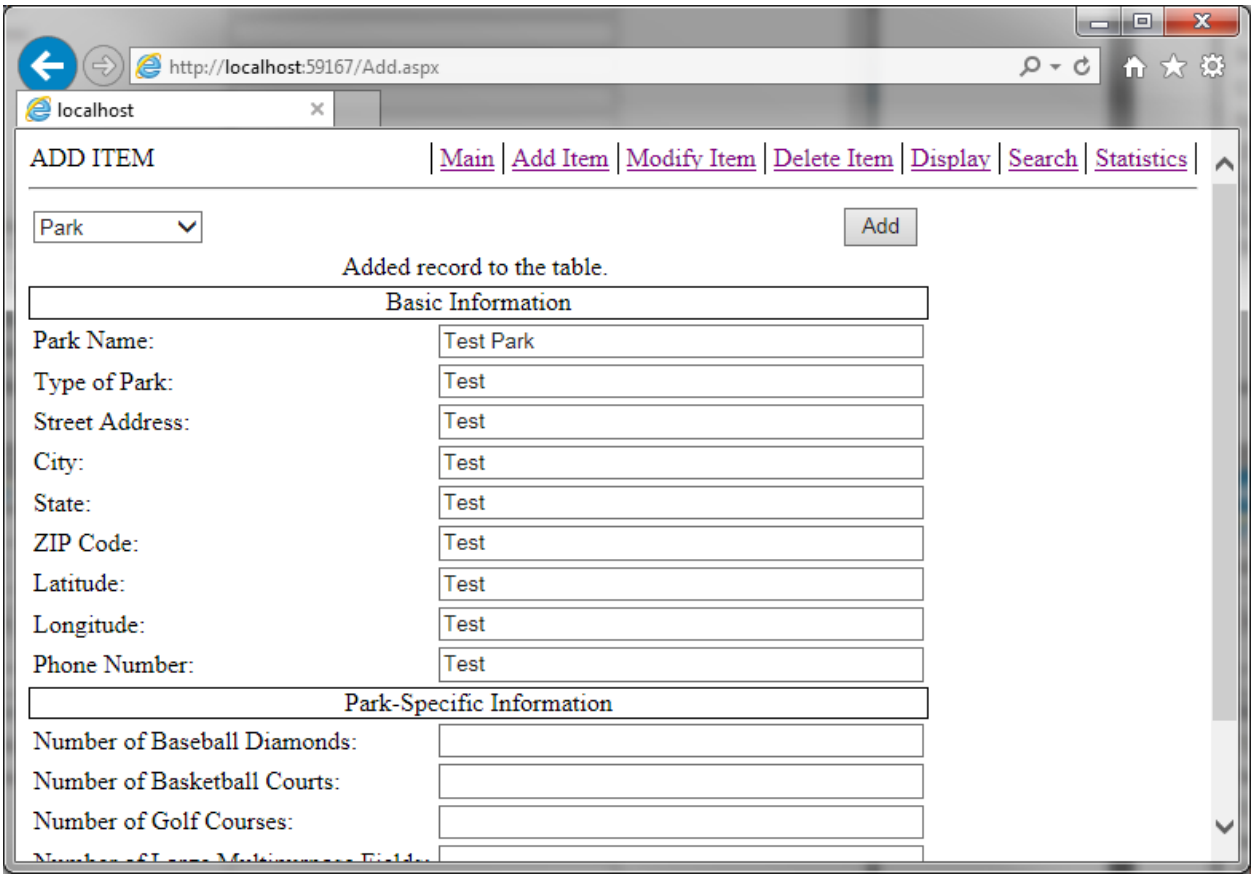
Attempting to add a public facility that already exists



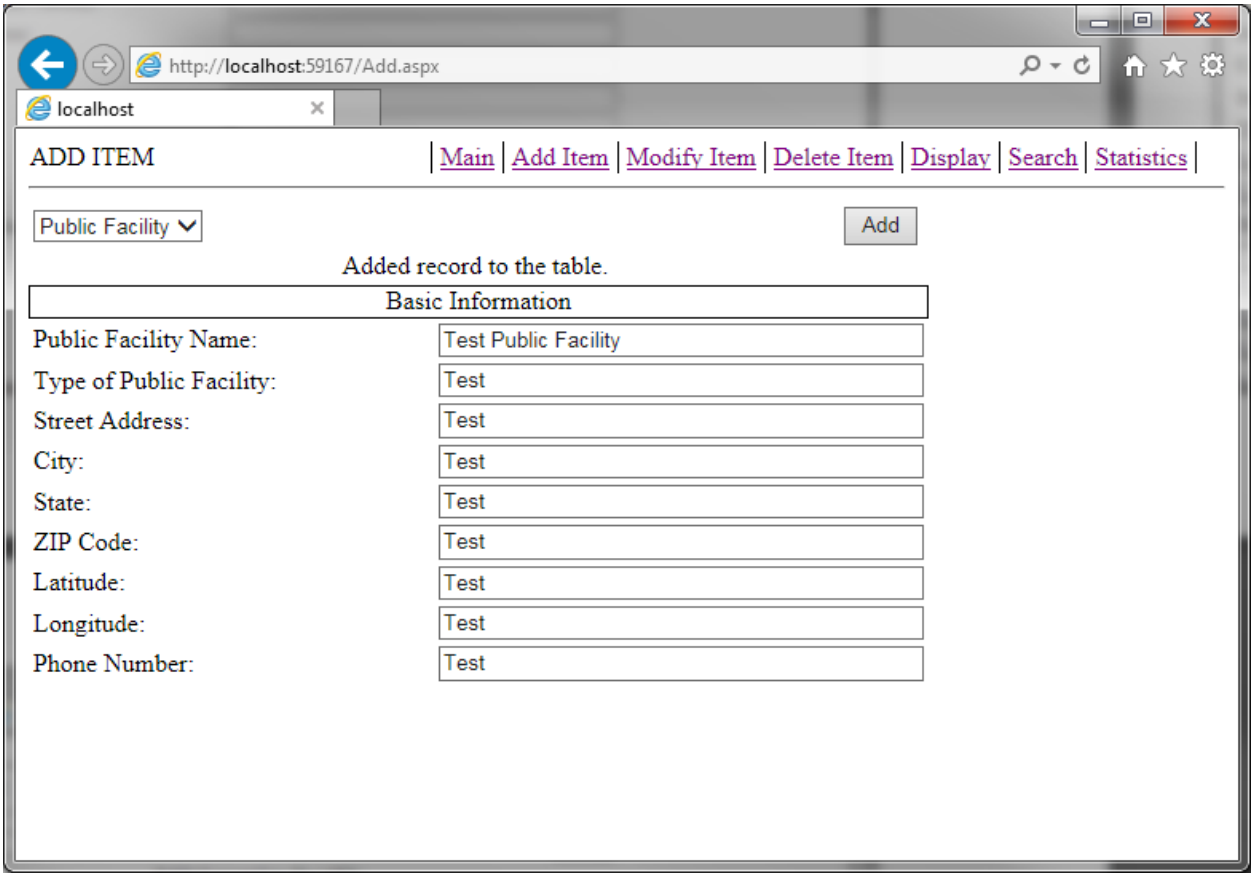
Successfully adding a business



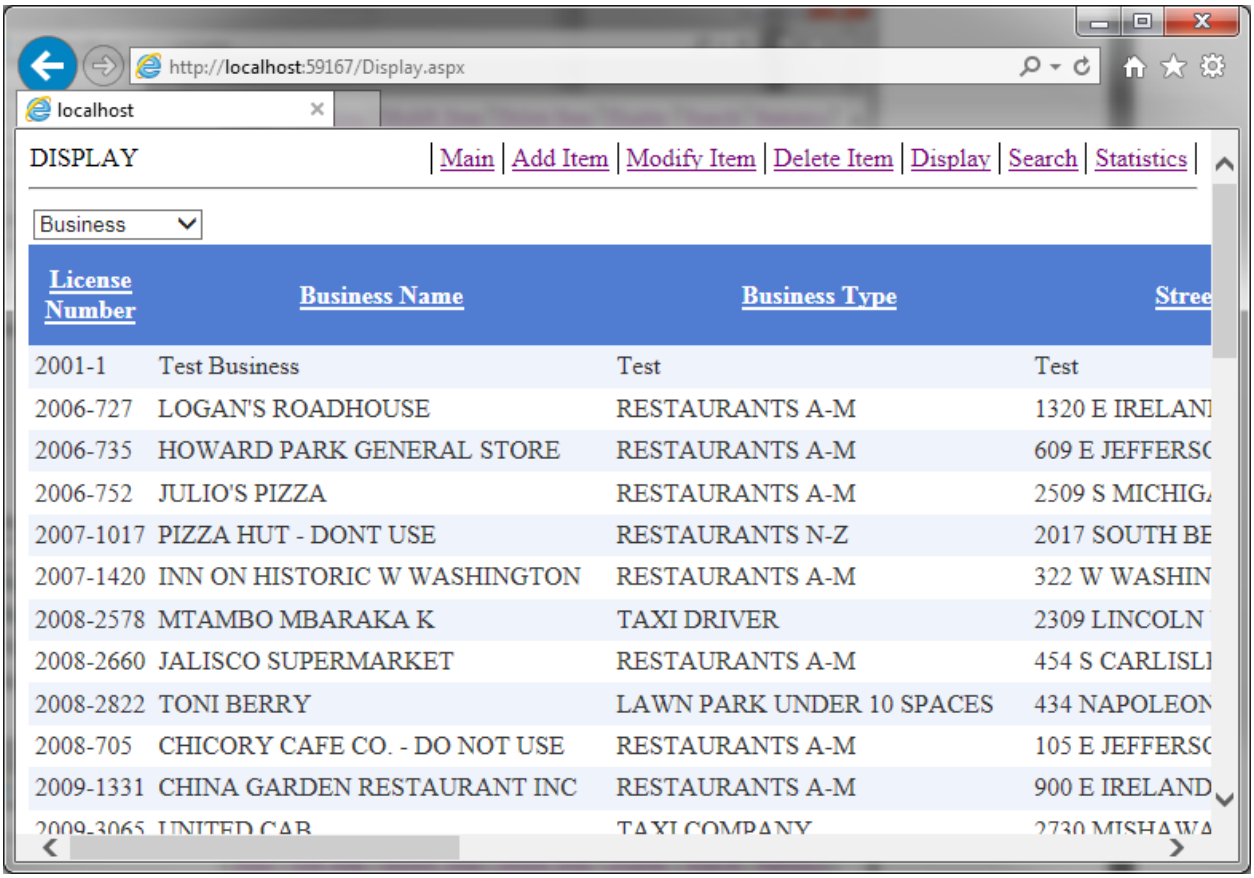
Successfully adding a park



Successfully adding a public facility



Displaying all businesses while noting that the test business is there



http://localhost:59167/Display.aspx

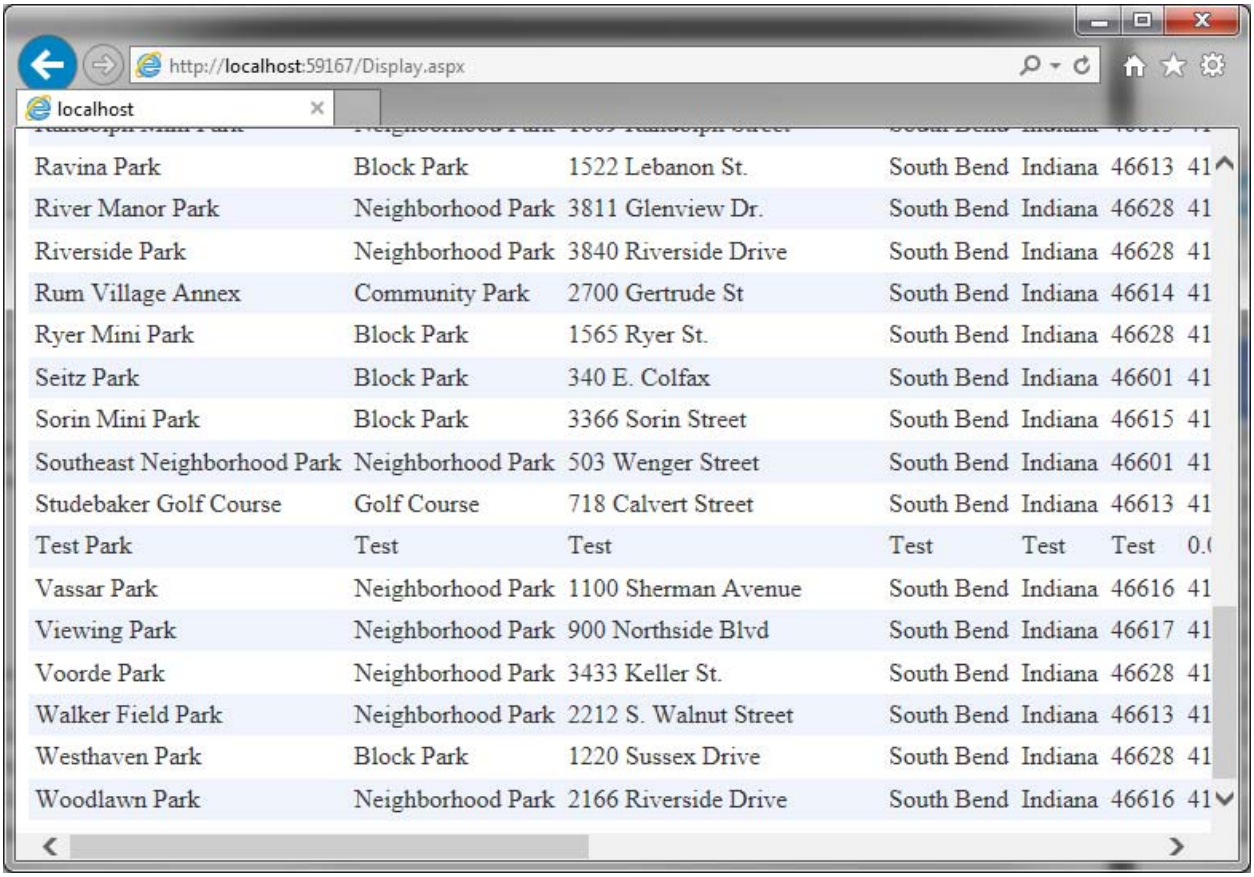
localhost

DISPLAY | Main | Add Item | Modify Item | Delete Item | Display | Search | Statistics |

Business

License Number	Business Name	Business Type	Stree
2001-1	Test Business	Test	Test
2006-727	LOGAN'S ROADHOUSE	RESTAURANTS A-M	1320 E IRELANI
2006-735	HOWARD PARK GENERAL STORE	RESTAURANTS A-M	609 E JEFFERSC
2006-752	JULIO'S PIZZA	RESTAURANTS A-M	2509 S MICHIG
2007-1017	PIZZA HUT - DONT USE	RESTAURANTS N-Z	2017 SOUTH BE
2007-1420	INN ON HISTORIC W WASHINGTON	RESTAURANTS A-M	322 W WASHIN
2008-2578	MTAMBO MBARAKA K	TAXI DRIVER	2309 LINCOLN
2008-2660	JALISCO SUPERMARKET	RESTAURANTS A-M	454 S CARLISLI
2008-2822	TONI BERRY	LAWN PARK UNDER 10 SPACES	434 NAPOLEON
2008-705	CHICORY CAFE CO. - DO NOT USE	RESTAURANTS A-M	105 E JEFFERSC
2009-1331	CHINA GARDEN RESTAURANT INC	RESTAURANTS A-M	900 E IRELAND
2009-3065	UNITED CAB	TAXI COMPANY	2730 MISHAWA

Displaying all parks while noting that the test park is there



http://localhost:59167/Display.aspx

localhost

Ravina Park	Block Park	1522 Lebanon St.	South Bend	Indiana	46613	41
River Manor Park	Neighborhood Park	3811 Glenview Dr.	South Bend	Indiana	46628	41
Riverside Park	Neighborhood Park	3840 Riverside Drive	South Bend	Indiana	46628	41
Rum Village Annex	Community Park	2700 Gertrude St	South Bend	Indiana	46614	41
Ryer Mini Park	Block Park	1565 Ryer St.	South Bend	Indiana	46628	41
Seitz Park	Block Park	340 E. Colfax	South Bend	Indiana	46601	41
Sorin Mini Park	Block Park	3366 Sorin Street	South Bend	Indiana	46615	41
Southeast Neighborhood Park	Neighborhood Park	503 Wenger Street	South Bend	Indiana	46601	41
Studebaker Golf Course	Golf Course	718 Calvert Street	South Bend	Indiana	46613	41
Test Park	Test	Test	Test	Test	Test	0.
Vassar Park	Neighborhood Park	1100 Sherman Avenue	South Bend	Indiana	46616	41
Viewing Park	Neighborhood Park	900 Northside Blvd	South Bend	Indiana	46617	41
Voorde Park	Neighborhood Park	3433 Keller St.	South Bend	Indiana	46628	41
Walker Field Park	Neighborhood Park	2212 S. Walnut Street	South Bend	Indiana	46613	41
Westhaven Park	Block Park	1220 Sussex Drive	South Bend	Indiana	46628	41
Woodlawn Park	Neighborhood Park	2166 Riverside Drive	South Bend	Indiana	46616	41

Displaying all public facilities while noting that the test public facility is there

http://localhost:59167/Display.aspx

localhost

South Bend Fire Dept. Station #9	FIRE STATION	2520 Mishawaka Ave.	Soutl
South Bend Police Department	POLICE STATION	101 W SAMPLE	Soutl
St. Joseph Co. Public Library Centre Township Branch	LIBRARY	1150 Kern Rd	Soutl
St. Joseph Co. Public Library Francis Branch	LIBRARY	52655 Ironwood Dr.	Soutl
St. Joseph Co. Public Library German Township Branch	LIBRARY	52807 Lynnewood Av	Soutl
St. Joseph Co. Public Library Lakeville Branch	LIBRARY	120 N Michigan St	Lake
St. Joseph Co. Public Library LaSalle Branch	LIBRARY	3232 Ardmore Tr.	Soutl
St. Joseph Co. Public Library Main Branch	LIBRARY	304 S. Main St.	Soutl
St. Joseph Co. Public Library North Liberty Branch	LIBRARY	105 E Market St	Nortl
St. Joseph Co. Public Library River Park Branch	LIBRARY	2022 Mishawaka Av.	Soutl
St. Joseph Co. Public Library Tutt Branch	LIBRARY	2223 Miami St.	Soutl
St. Joseph Co. Public Library Western Branch	LIBRARY	611 Lombardy Dr	Soutl
St. Joseph County Jail	POLICE STATION	701 W SAMPLE	Soutl
Test Public Facility	Test	Test	Test
Warren Twp. Sta. 1	FIRE STATION	54837 Quince Road	Soutl
Warren Twp. Sta. 2	FIRE STATION	59330 Crumstown Hwy.	Soutl

Selecting the test business for modification

http://localhost:59167/Modify.aspx

localhost

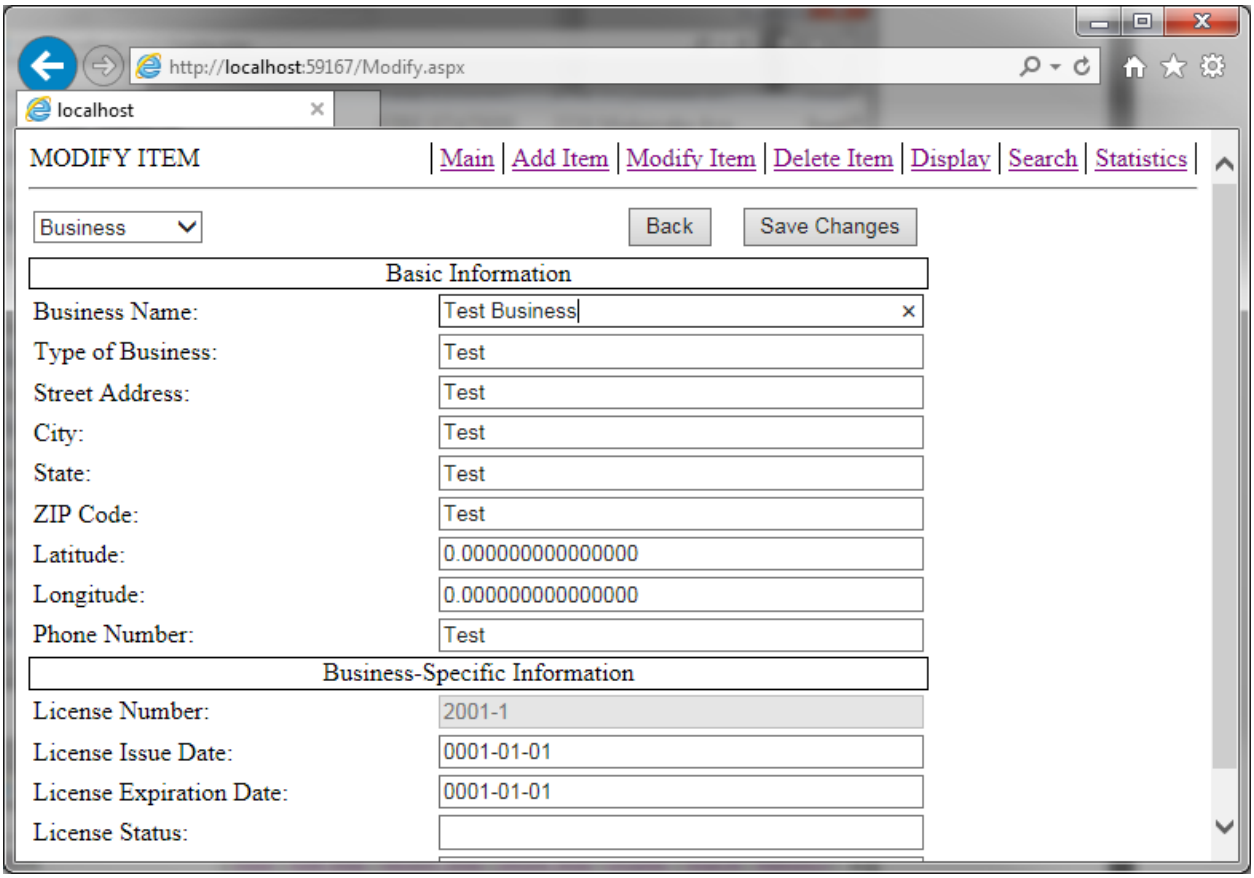
MODIFY ITEM | Main | Add Item | Modify Item | Delete Item | Display | Search | Statistics |

Business

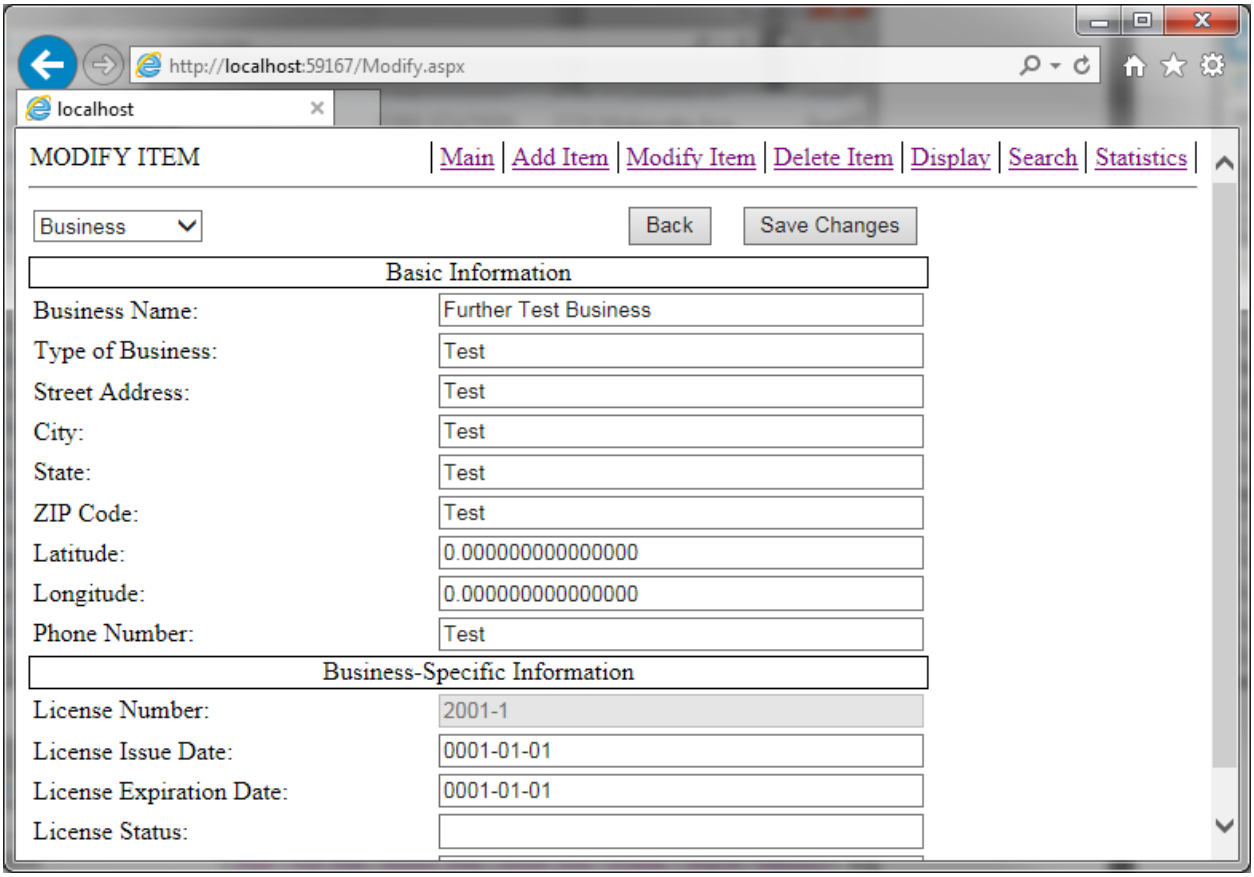
Modify

	License Number	Business Name	Street Address
Select	**2001-1**	**Test Business**	**Test**
Select	2006-727	LOGAN'S ROADHOUSE	1320 E IRELAND RD
Select	2006-735	HOWARD PARK GENERAL STORE	609 E JEFFERSON BLVD
Select	2006-752	JULIO'S PIZZA	2509 S MICHIGAN ST
Select	2007-1017	PIZZA HUT - DONT USE	2017 SOUTH BEND AVE
Select	2007-1420	INN ON HISTORIC W WASHINGTON	322 W WASHINGTON ST
Select	2008-2578	MTAMBO MBARAKA K	2309 LINCOLN WAY W
Select	2008-2660	JALISCO SUPERMARKET	454 S CARLISLE ST
Select	2008-2822	TONI BERRY	434 NAPOLEON ST
Select	2008-705	CHICORY CAFE CO. - DO NOT USE	105 E JEFFERSON BLVD
Select	2009-1331	CHINA GARDEN RESTAURANT INC	900 E IRELAND RD
Select	2009-3065	UNITED CAB	2730 MISHAWAKA AVE
Select	2009-3137	ABC CAB & COURIER EXPRESS CAB	1733 S MICHIGAN ST

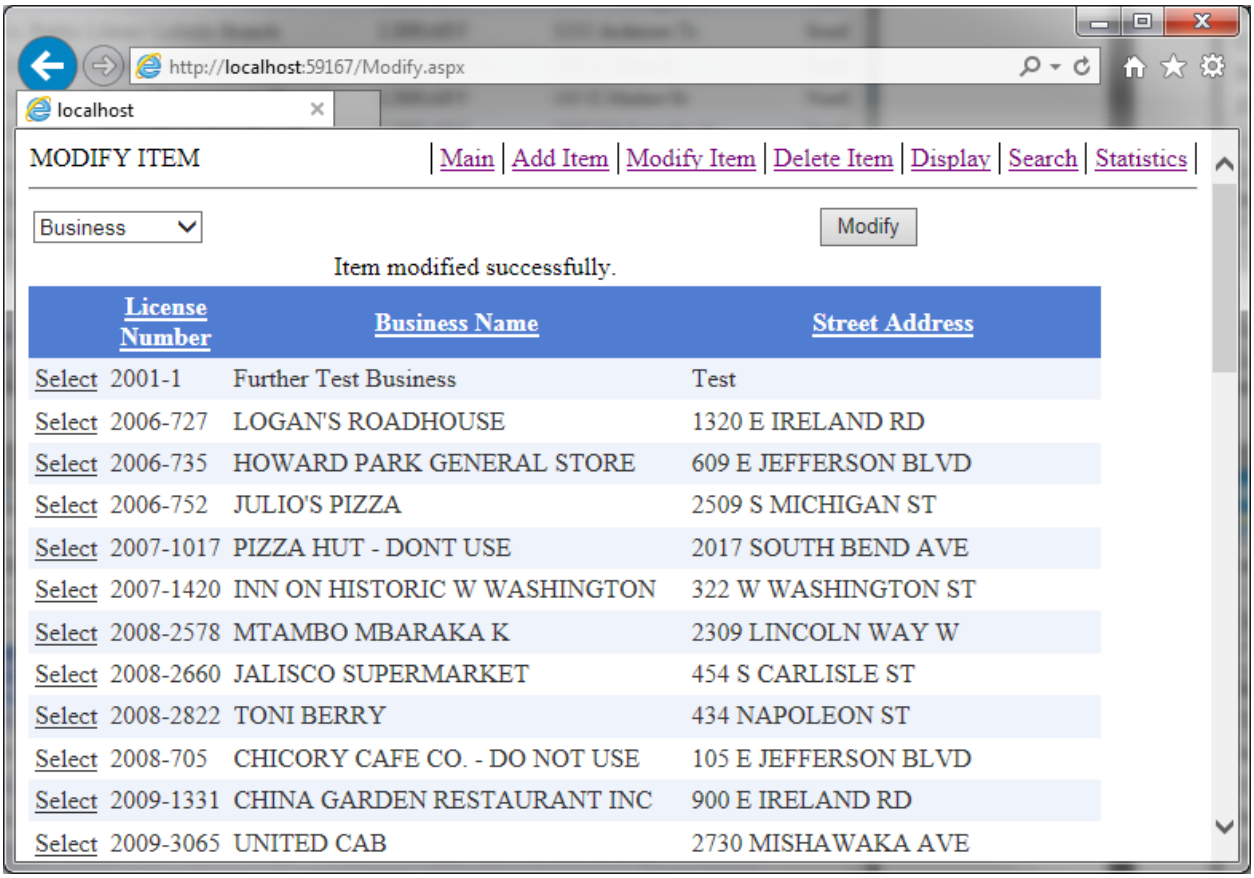
Observing that the text boxes are filled with the information for the test business



Showing the modification to be done to the test business

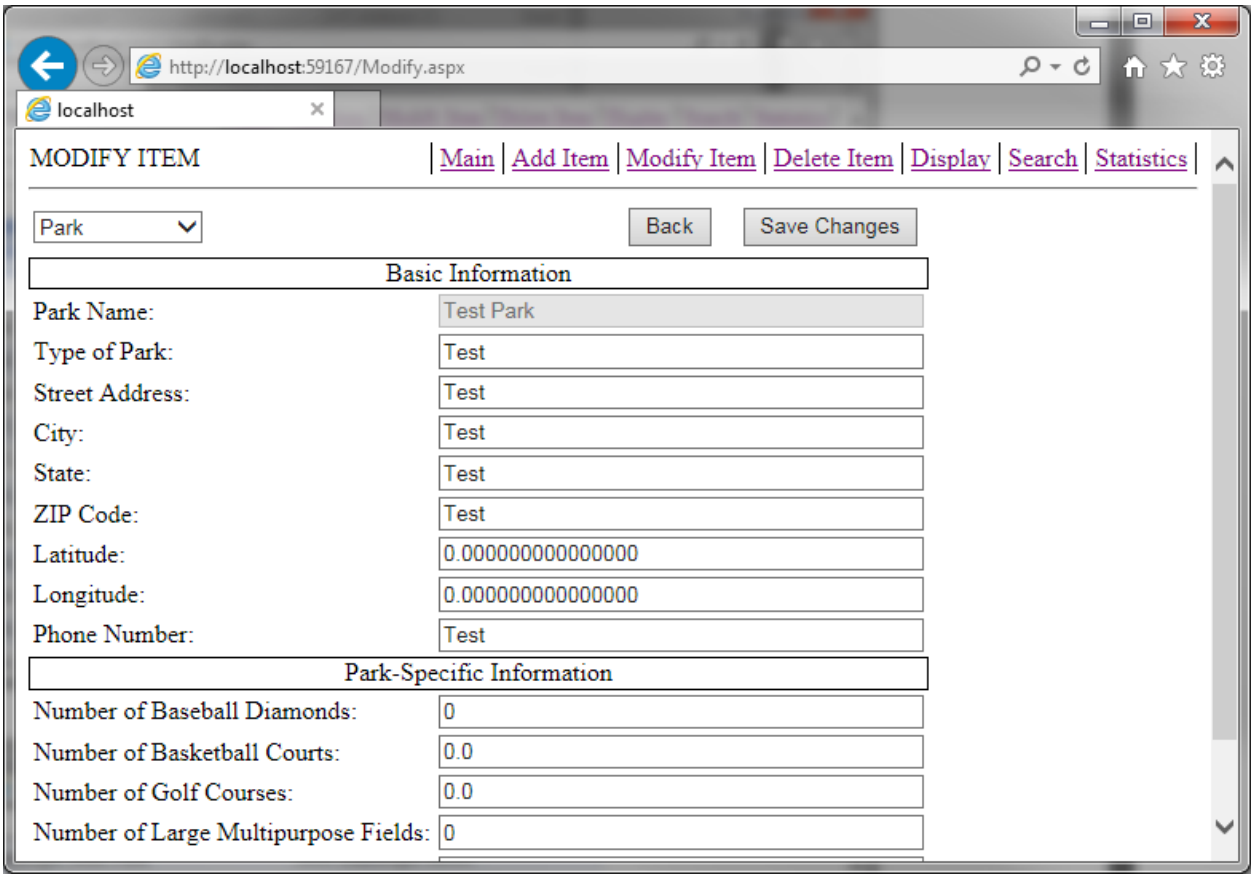


Showing the completed modification to the test business

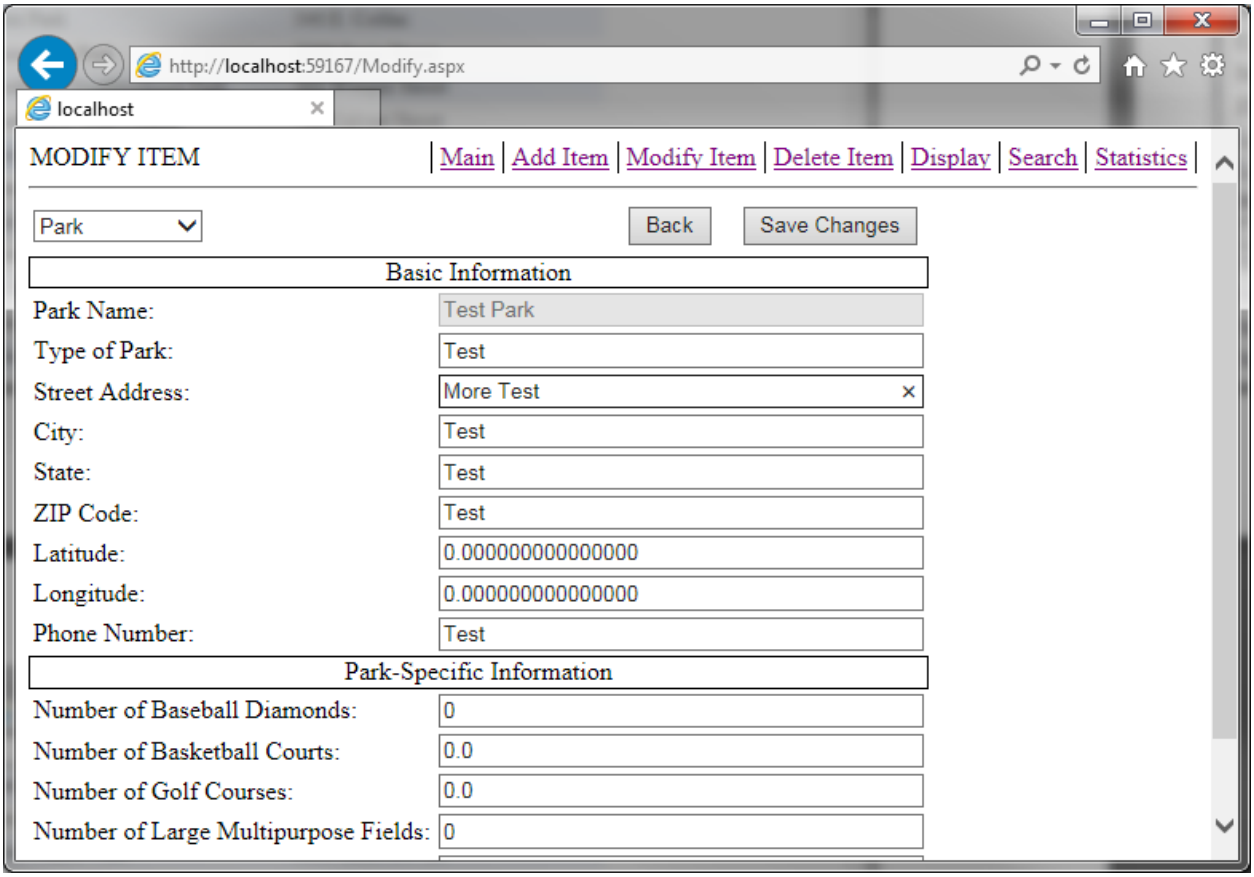


Selecting the test park for modification

Observing that the text boxes are filled with the information for the test park



Showing the modification to be done to the test park



Showing the completed modification to the test park

http://localhost:59167/Modify.aspx

localhost

Select	Randolph Mini Park	1809 Randolph Street
Select	Ravina Park	1522 Lebanon St.
Select	River Manor Park	3811 Glenview Dr.
Select	Riverside Park	3840 Riverside Drive
Select	Rum Village Annex	2700 Gertrude St
Select	Ryer Mini Park	1565 Ryer St.
Select	Seitz Park	340 E. Colfax
Select	Sorin Mini Park	3366 Sorin Street
Select	Southeast Neighborhood Park	503 Wenger Street
Select	Studebaker Golf Course	718 Calvert Street
Select	Test Park	More Test
Select	Vassar Park	1100 Sherman Avenue
Select	Viewing Park	900 Northside Blvd
Select	Voorde Park	3433 Keller St.
Select	Walker Field Park	2212 S. Walnut Street
Select	Westhaven Park	1220 Sussex Drive
Select	Woodlawn Park	2166 Riverside Drive

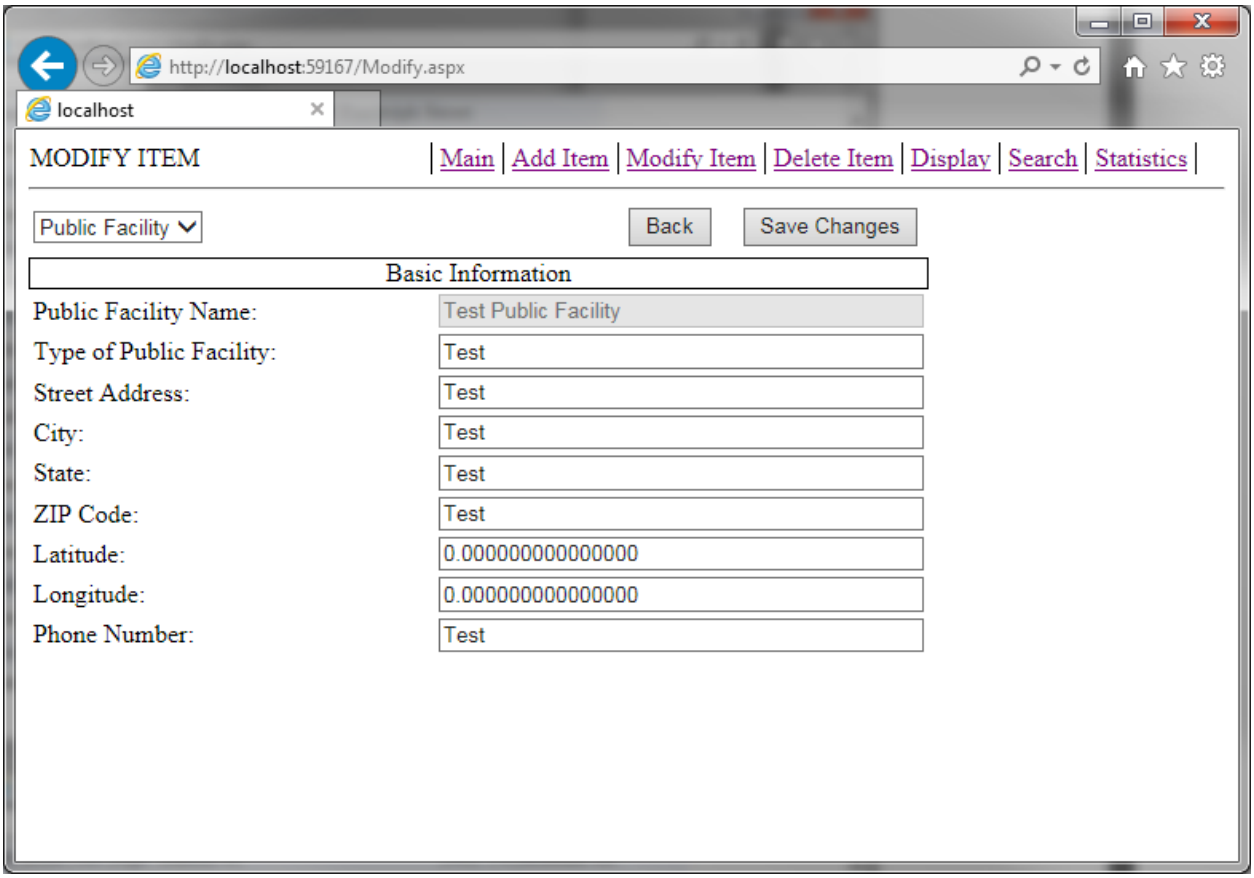
Selecting the test public facility for modification

http://localhost:59167/Modify.aspx

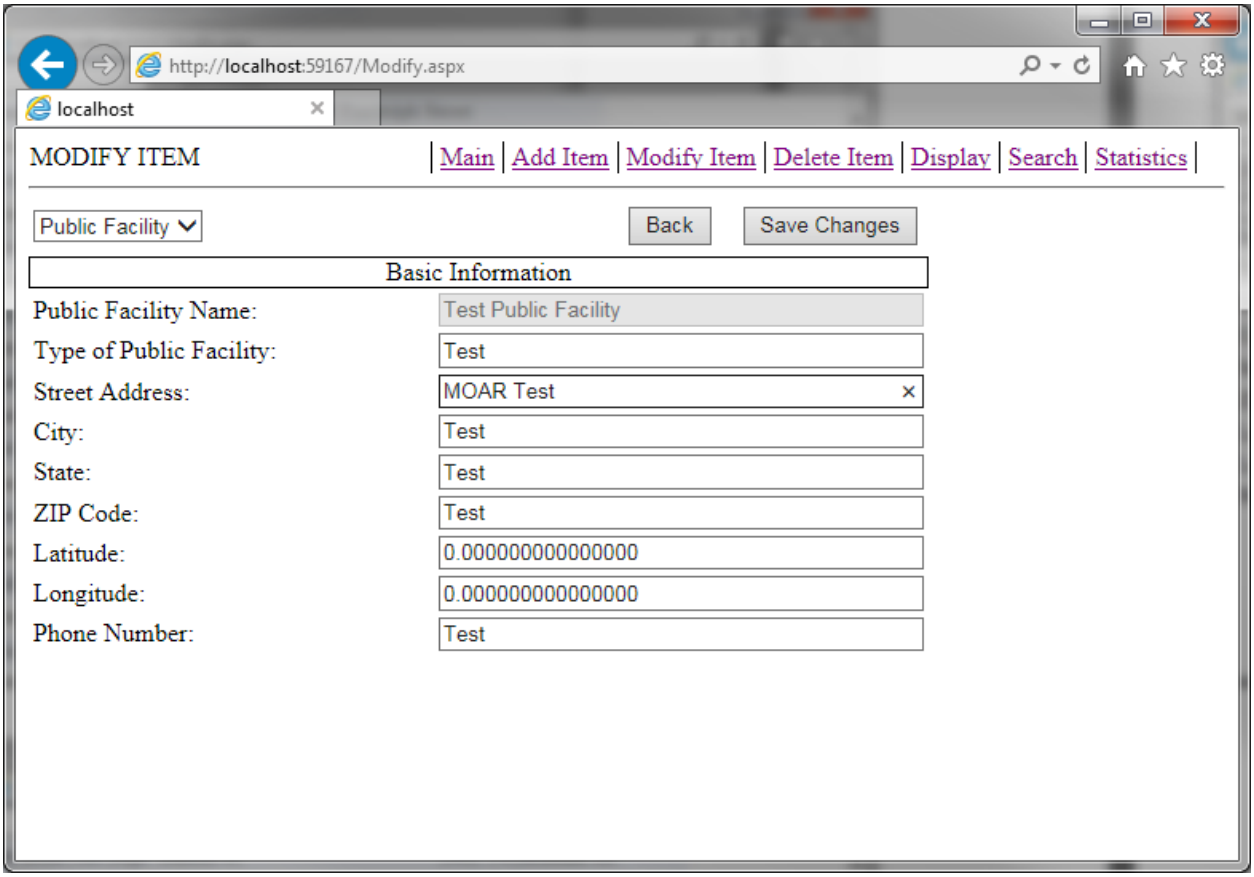
localhost

Select	South Bend Fire Dept. Station #8	2402 Twykenham Dr.
Select	South Bend Fire Dept. Station #9	2520 Mishawaka Ave.
Select	South Bend Police Department	101 W SAMPLE
Select	St. Joseph Co. Public Library Centre Township Branch	1150 Kern Rd
Select	St. Joseph Co. Public Library Francis Branch	52655 Ironwood Dr.
Select	St. Joseph Co. Public Library German Township Branch	52807 Lynnewood Av
Select	St. Joseph Co. Public Library Lakeville Branch	120 N Michigan St
Select	St. Joseph Co. Public Library LaSalle Branch	3232 Ardmore Tr.
Select	St. Joseph Co. Public Library Main Branch	304 S. Main St.
Select	St. Joseph Co. Public Library North Liberty Branch	105 E Market St
Select	St. Joseph Co. Public Library River Park Branch	2022 Mishawaka Av.
Select	St. Joseph Co. Public Library Tutt Branch	2223 Miami St.
Select	St. Joseph Co. Public Library Western Branch	611 Lombardy Dr
Select	St. Joseph County Jail	701 W SAMPLE
Select	**Test Public Facility**	**Test**
Select	Warren Twp. Sta. 1	54837 Quince Road
Select	Warren Twp. Sta. 2	59330 Crumstown Hwy.

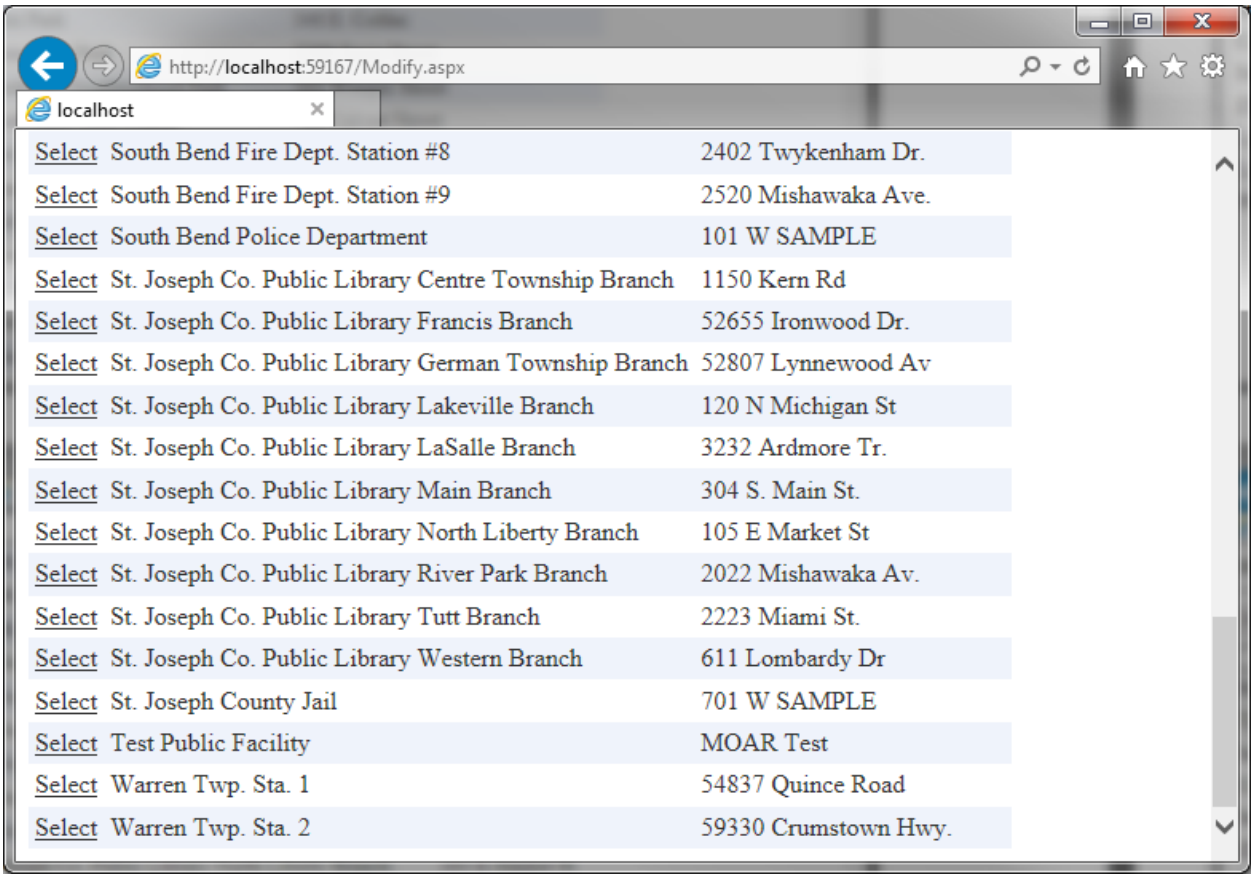
Observing that the text boxes are filled with the information for the test public facility



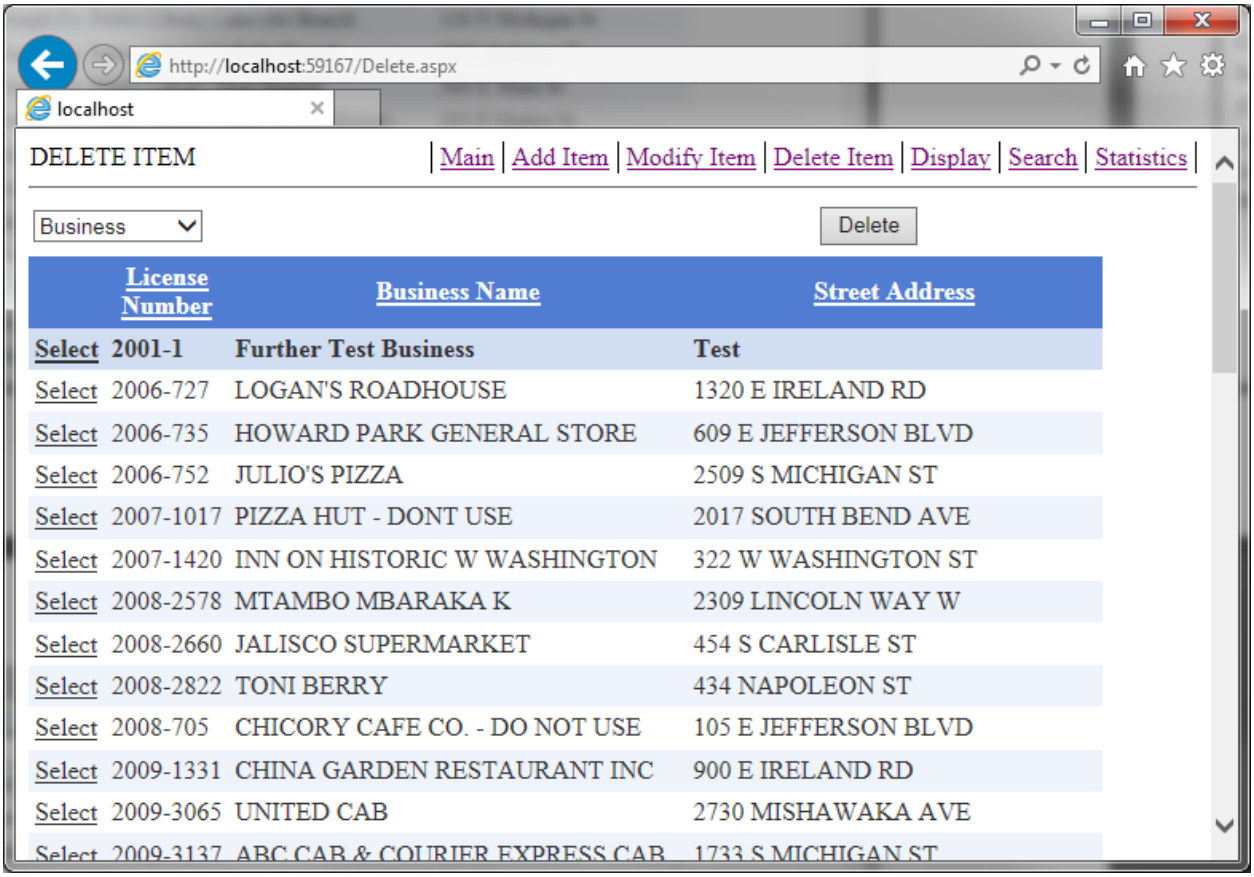
Showing the modification to be done to the test public facility



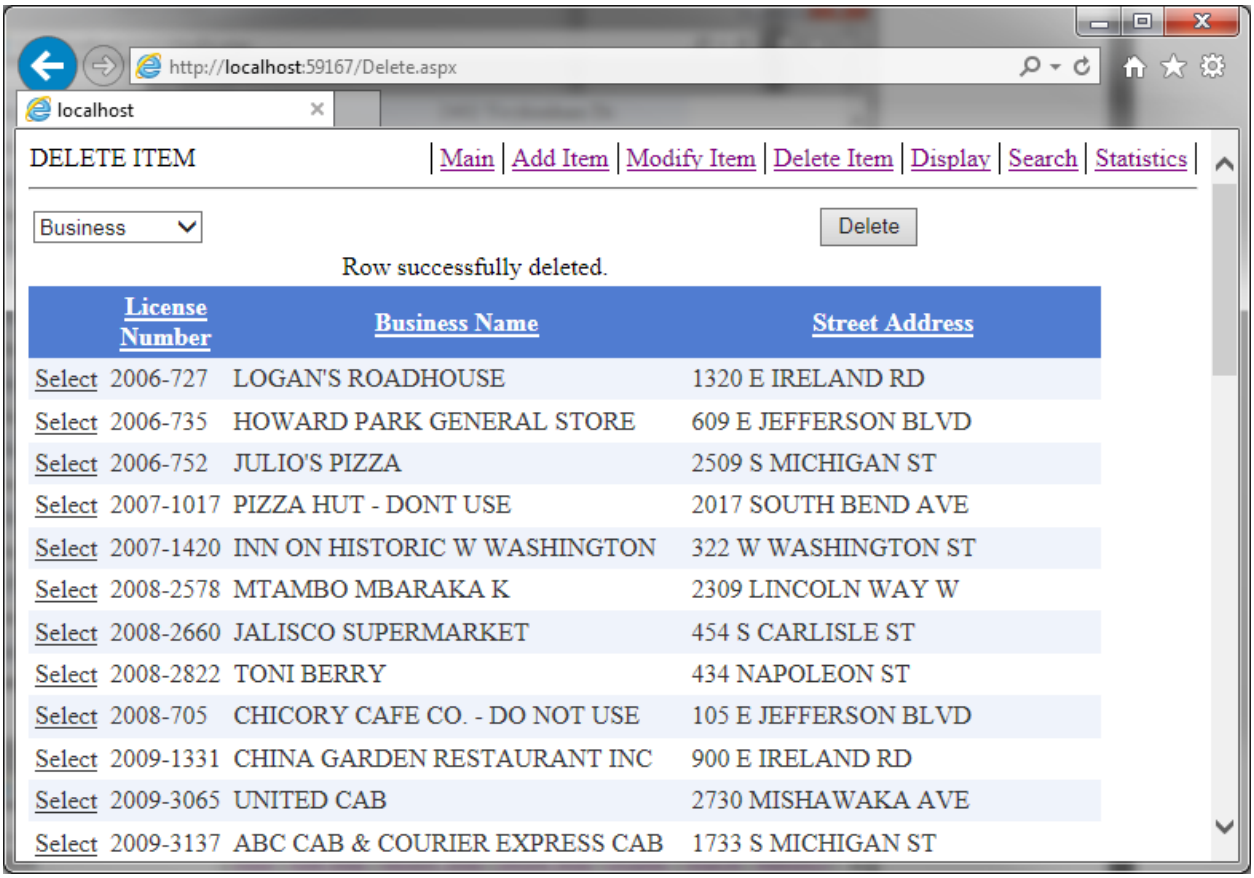
Showing the completed modification to the test public facility



Selecting the test business for deletion

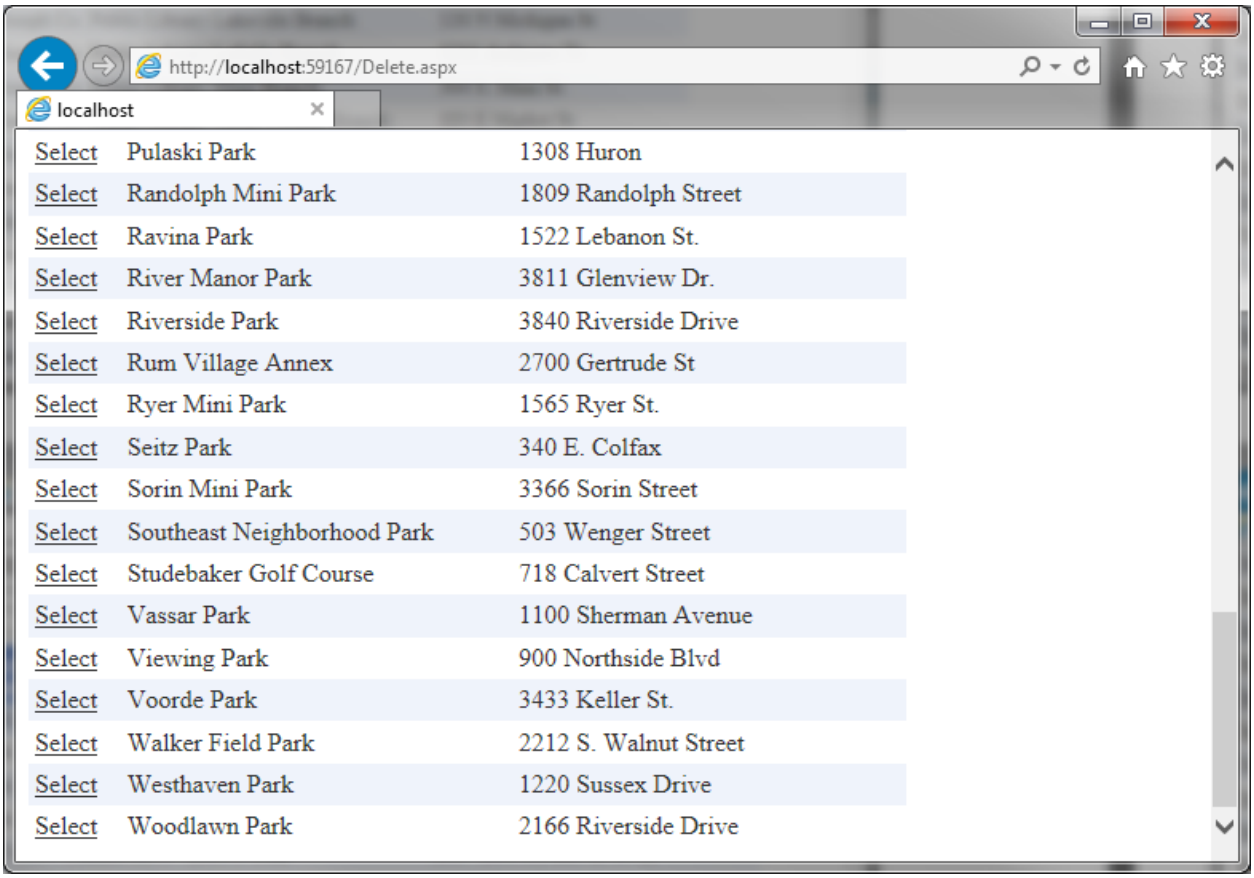


Showing that the test business has been deleted



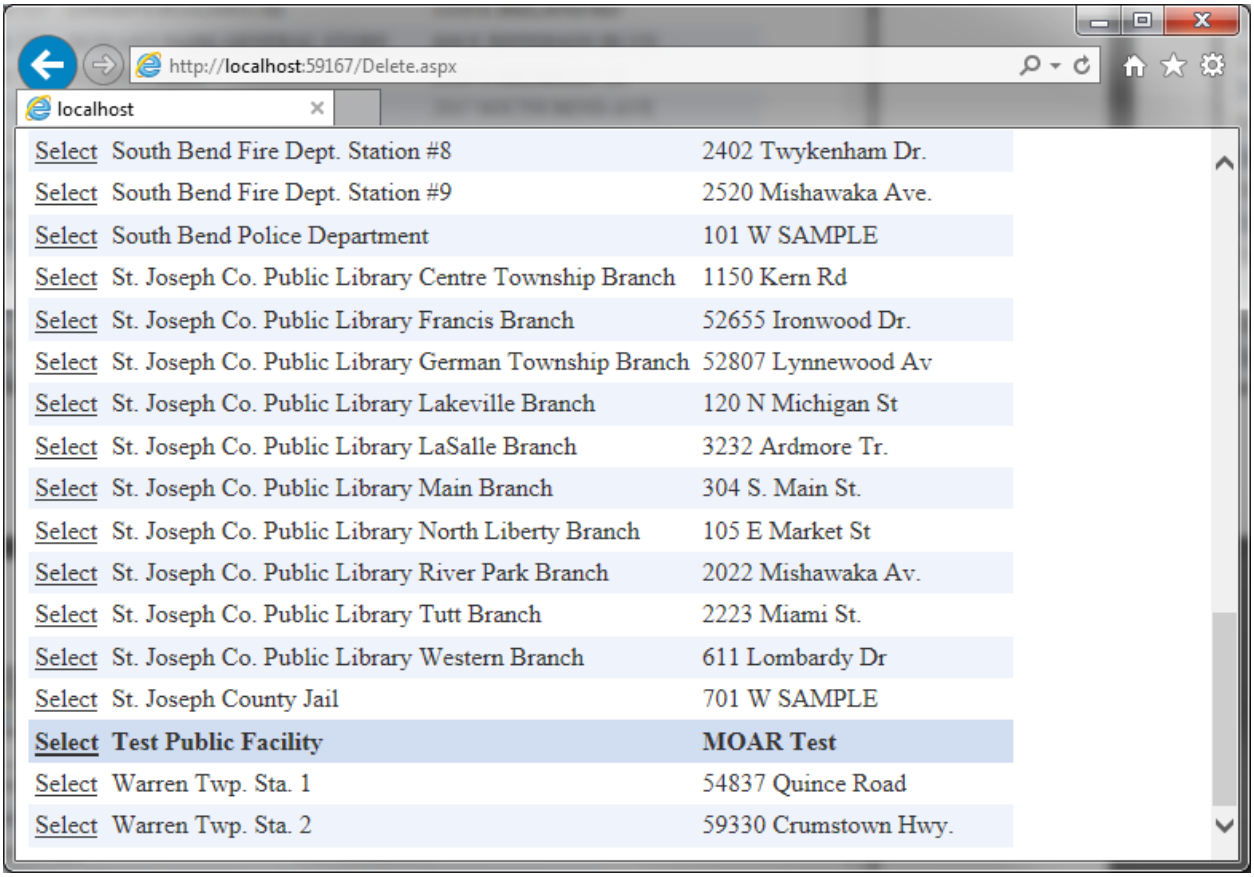
Selecting the test park for deletion

Showing that the test park has been deleted



Select	Pulaski Park	1308 Huron
Select	Randolph Mini Park	1809 Randolph Street
Select	Ravina Park	1522 Lebanon St.
Select	River Manor Park	3811 Glenview Dr.
Select	Riverside Park	3840 Riverside Drive
Select	Rum Village Annex	2700 Gertrude St
Select	Ryer Mini Park	1565 Ryer St.
Select	Seitz Park	340 E. Colfax
Select	Sorin Mini Park	3366 Sorin Street
Select	Southeast Neighborhood Park	503 Wenger Street
Select	Studebaker Golf Course	718 Calvert Street
Select	Vassar Park	1100 Sherman Avenue
Select	Viewing Park	900 Northside Blvd
Select	Voorde Park	3433 Keller St.
Select	Walker Field Park	2212 S. Walnut Street
Select	Westhaven Park	1220 Sussex Drive
Select	Woodlawn Park	2166 Riverside Drive

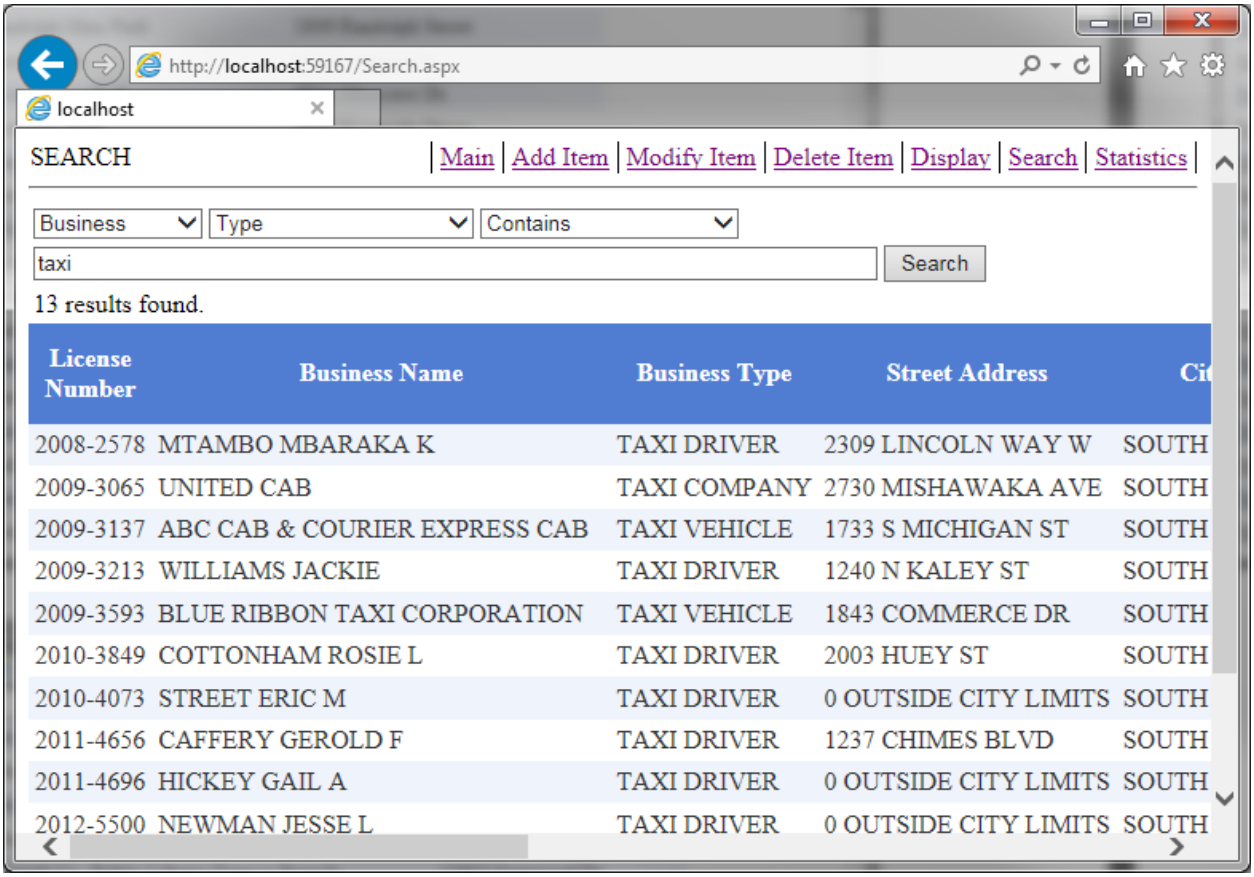
Selecting the test public facility for deletion



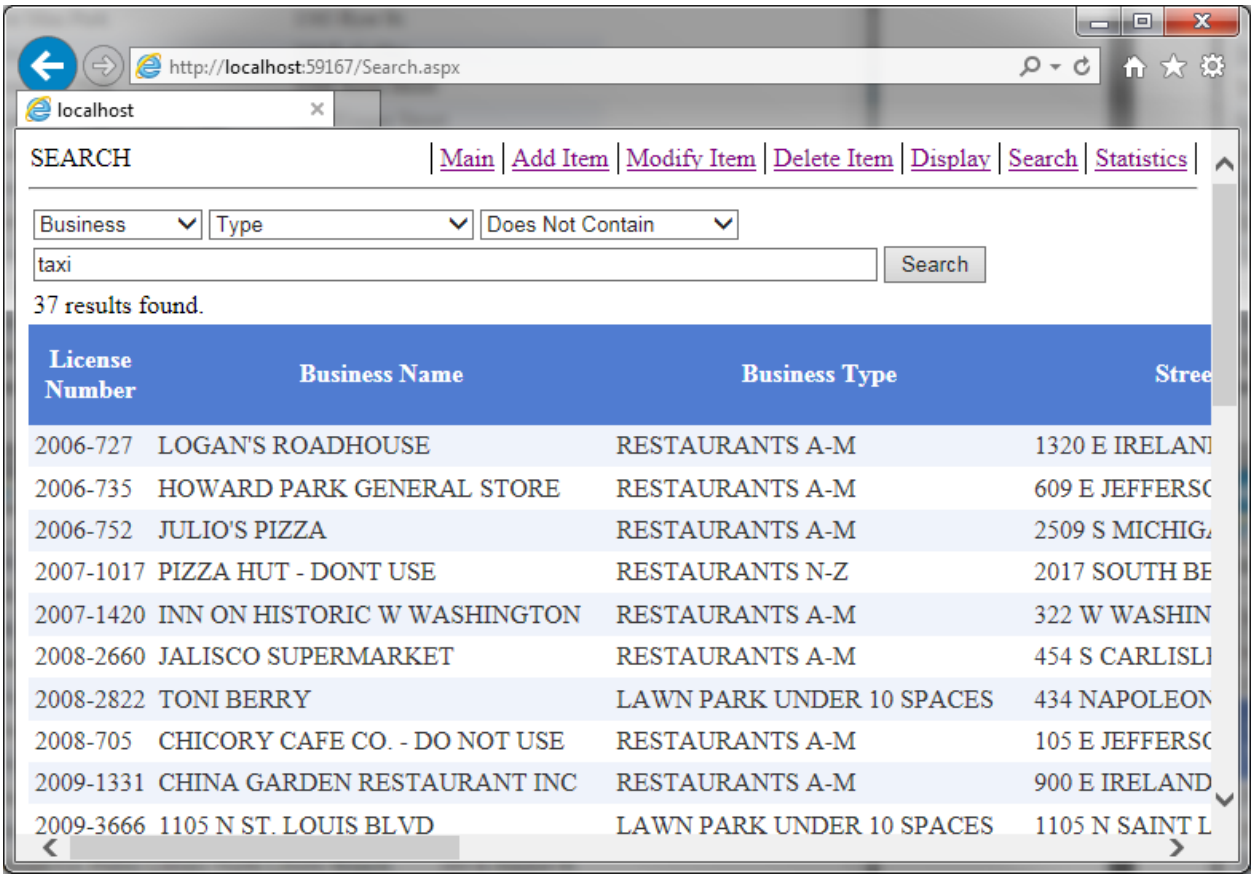
Select	South Bend Fire Dept. Station #8	2402 Twykenham Dr.
Select	South Bend Fire Dept. Station #9	2520 Mishawaka Ave.
Select	South Bend Police Department	101 W SAMPLE
Select	St. Joseph Co. Public Library Centre Township Branch	1150 Kern Rd
Select	St. Joseph Co. Public Library Francis Branch	52655 Ironwood Dr.
Select	St. Joseph Co. Public Library German Township Branch	52807 Lynnewood Av
Select	St. Joseph Co. Public Library Lakeville Branch	120 N Michigan St
Select	St. Joseph Co. Public Library LaSalle Branch	3232 Ardmore Tr.
Select	St. Joseph Co. Public Library Main Branch	304 S. Main St.
Select	St. Joseph Co. Public Library North Liberty Branch	105 E Market St
Select	St. Joseph Co. Public Library River Park Branch	2022 Mishawaka Av.
Select	St. Joseph Co. Public Library Tutt Branch	2223 Miami St.
Select	St. Joseph Co. Public Library Western Branch	611 Lombardy Dr
Select	St. Joseph County Jail	701 W SAMPLE
Select	**Test Public Facility**	**MOAR Test**
Select	Warren Twp. Sta. 1	54837 Quince Road
Select	Warren Twp. Sta. 2	59330 Crumstown Hwy.

Showing that the test public facility has been deleted

Demonstrating various searches

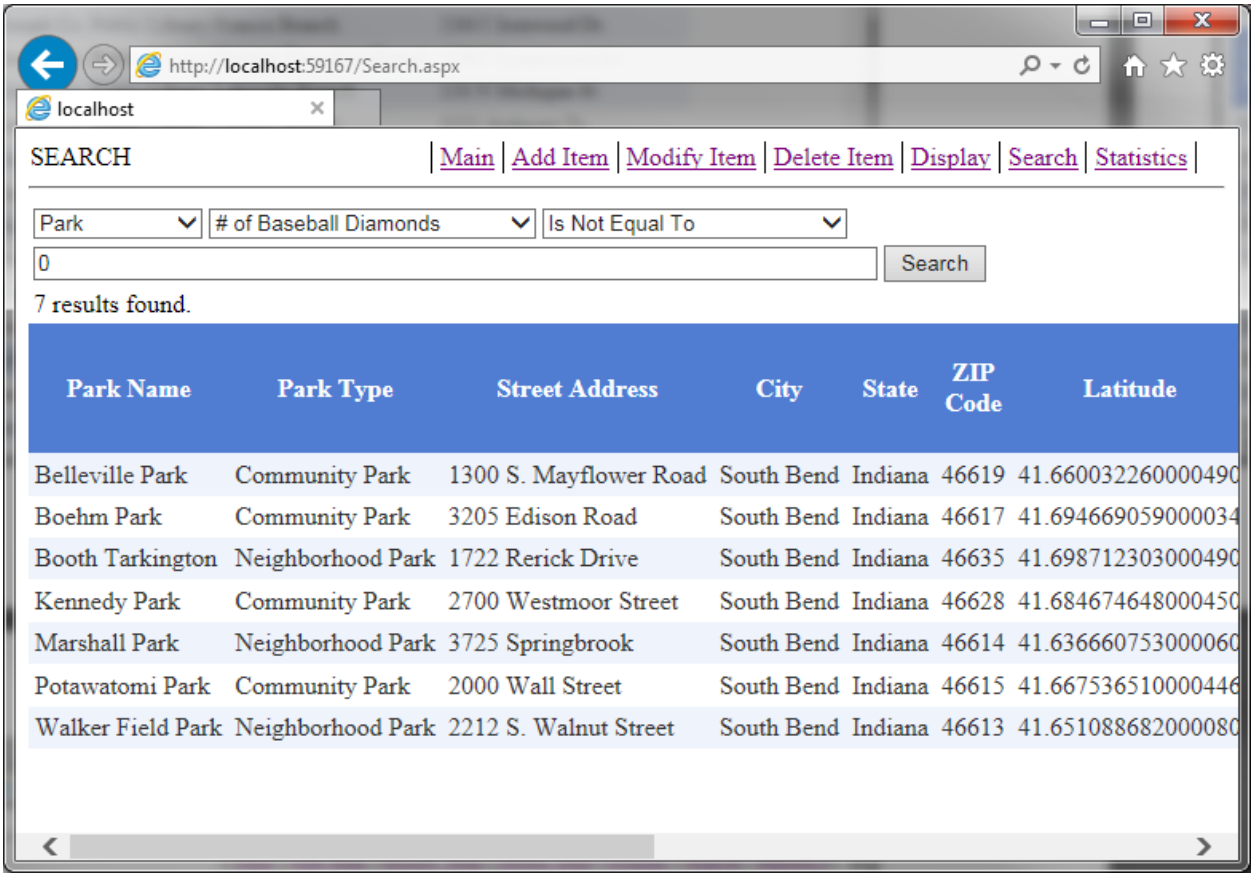


Demonstrating various searches



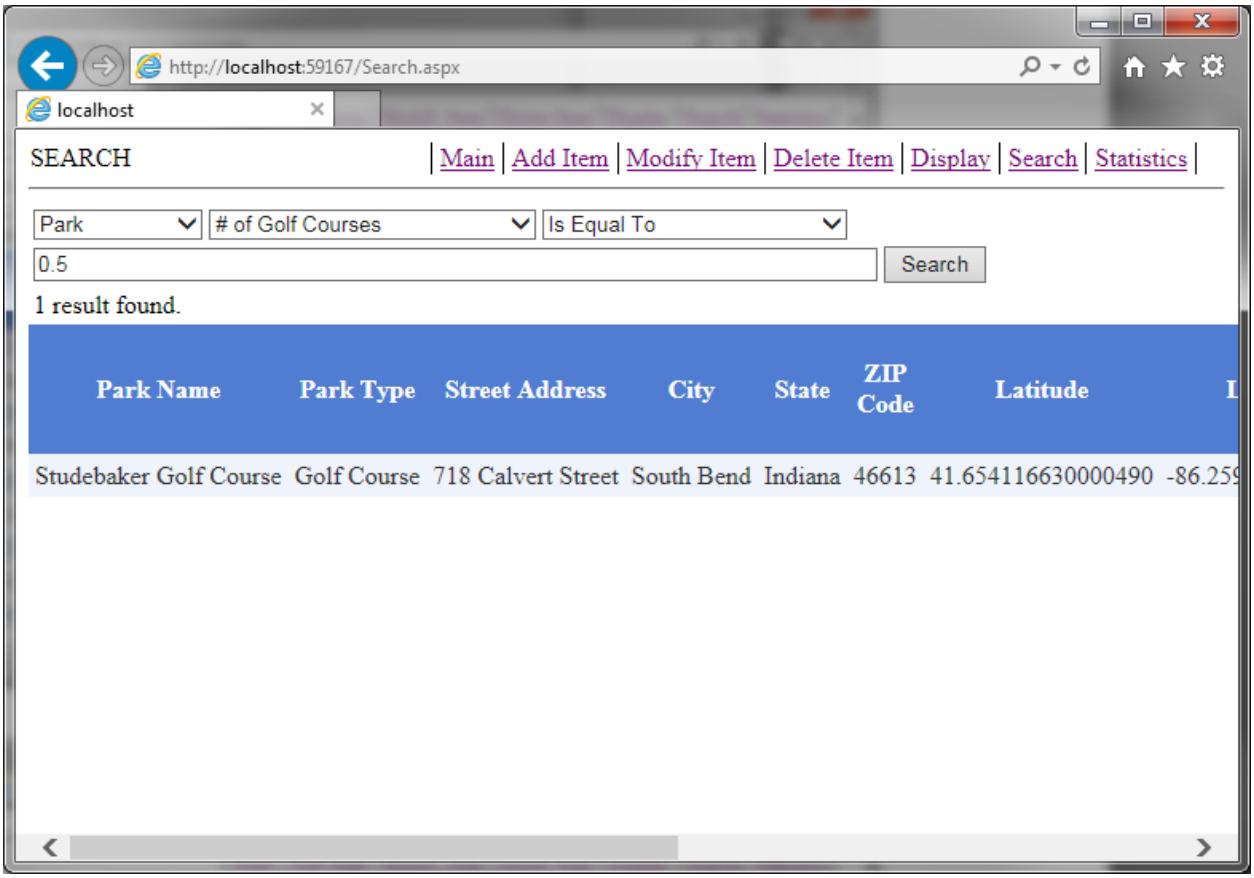
License Number	Business Name	Business Type	Stree
2006-727	LOGAN'S ROADHOUSE	RESTAURANTS A-M	1320 E IRELANI
2006-735	HOWARD PARK GENERAL STORE	RESTAURANTS A-M	609 E JEFFERSC
2006-752	JULIO'S PIZZA	RESTAURANTS A-M	2509 S MICHIGA
2007-1017	PIZZA HUT - DONT USE	RESTAURANTS N-Z	2017 SOUTH BE
2007-1420	INN ON HISTORIC W WASHINGTON	RESTAURANTS A-M	322 W WASHIN
2008-2660	JALISCO SUPERMARKET	RESTAURANTS A-M	454 S CARLISLI
2008-2822	TONI BERRY	LAWN PARK UNDER 10 SPACES	434 NAPOLEON
2008-705	CHICORY CAFE CO. - DO NOT USE	RESTAURANTS A-M	105 E JEFFERSC
2009-1331	CHINA GARDEN RESTAURANT INC	RESTAURANTS A-M	900 E IRELAND
2009-3666	1105 N ST. LOUIS BLVD	LAWN PARK UNDER 10 SPACES	1105 N SAINT L

Demonstrating various searches

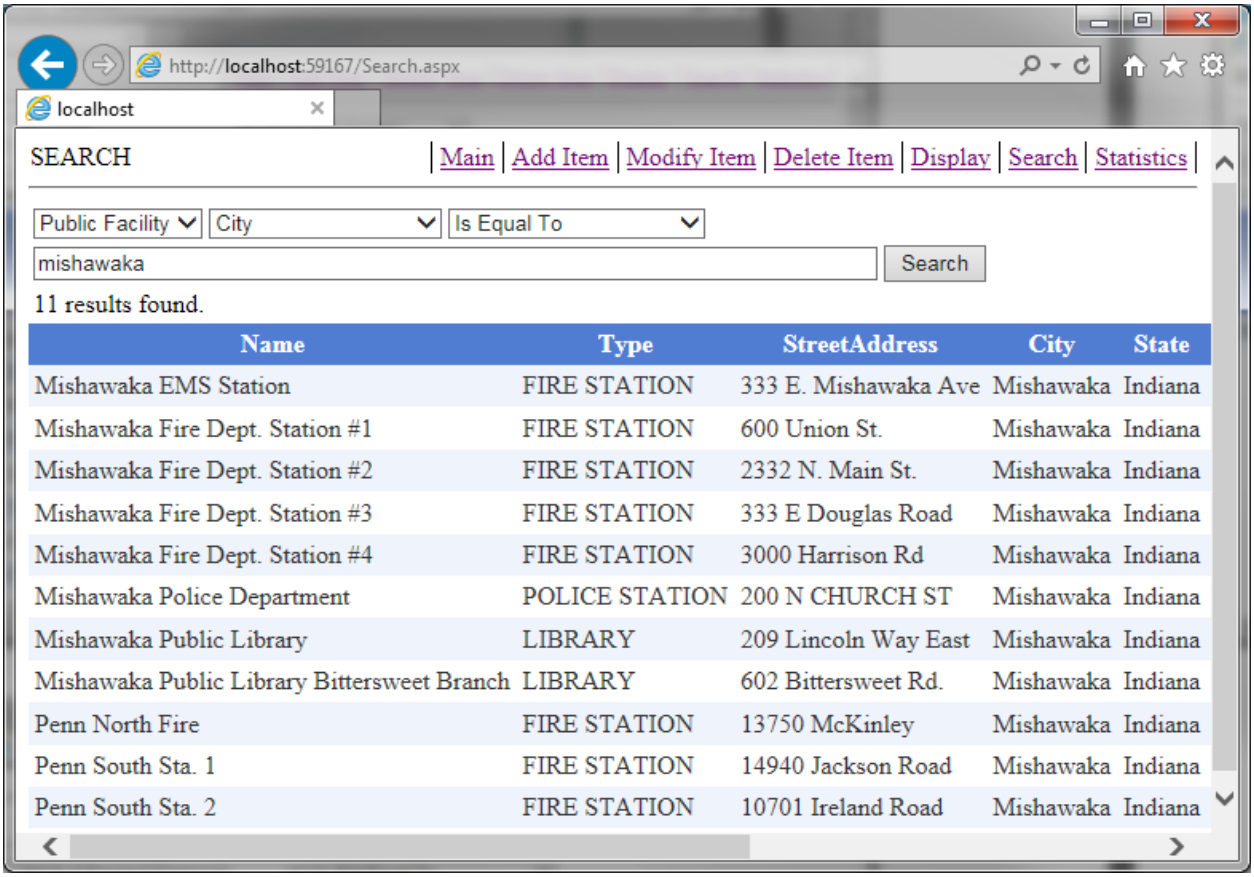


Park Name	Park Type	Street Address	City	State	ZIP Code	Latitude
Belleville Park	Community Park	1300 S. Mayflower Road	South Bend	Indiana	46619	41.660032260000490
Boehm Park	Community Park	3205 Edison Road	South Bend	Indiana	46617	41.694669059000034
Booth Tarkington	Neighborhood Park	1722 Rerick Drive	South Bend	Indiana	46635	41.698712303000490
Kennedy Park	Community Park	2700 Westmoor Street	South Bend	Indiana	46628	41.684674648000450
Marshall Park	Neighborhood Park	3725 Springbrook	South Bend	Indiana	46614	41.636660753000060
Potawatomi Park	Community Park	2000 Wall Street	South Bend	Indiana	46615	41.667536510000446
Walker Field Park	Neighborhood Park	2212 S. Walnut Street	South Bend	Indiana	46613	41.651088682000080

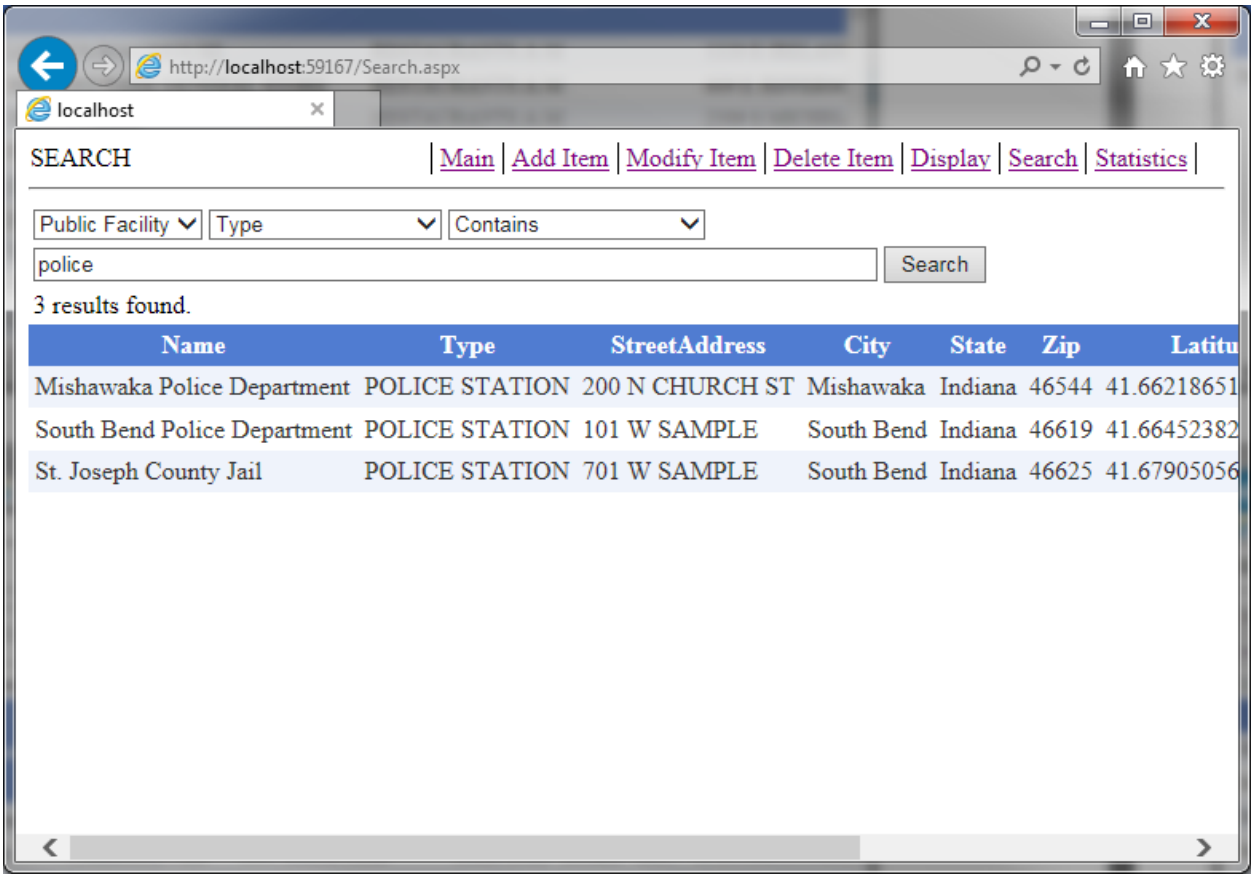
Demonstrating various searches



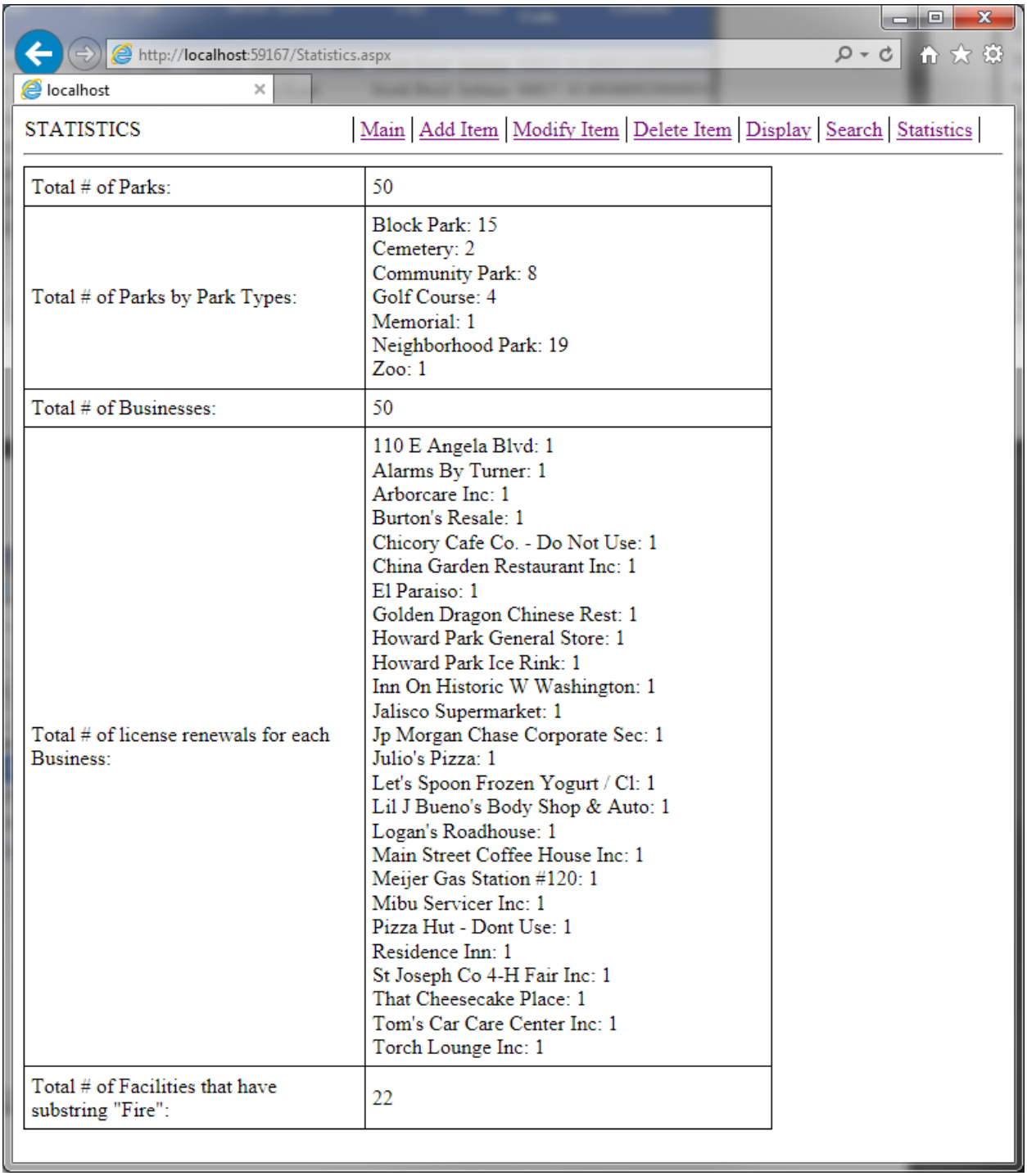
Demonstrating various searches



Demonstrating various searches

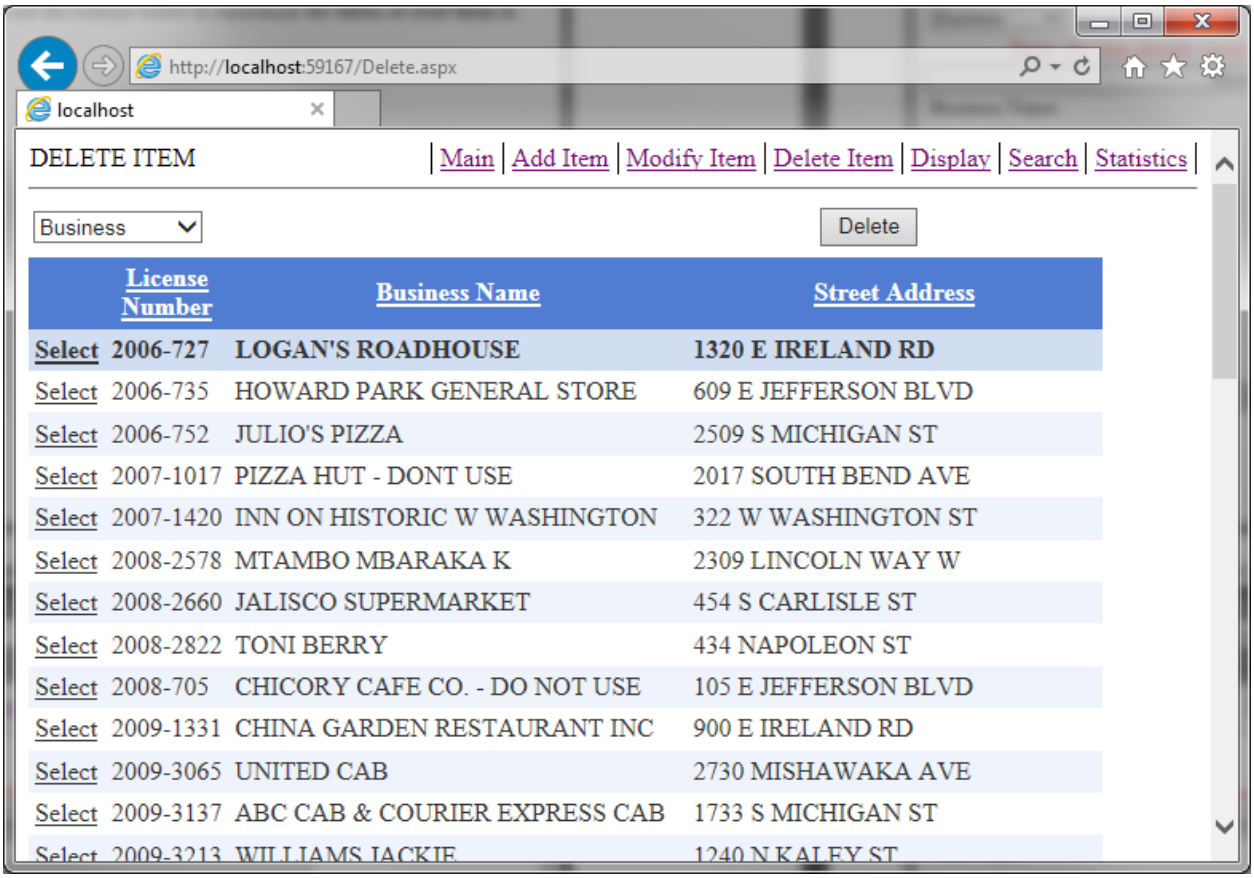


Showing the statistics

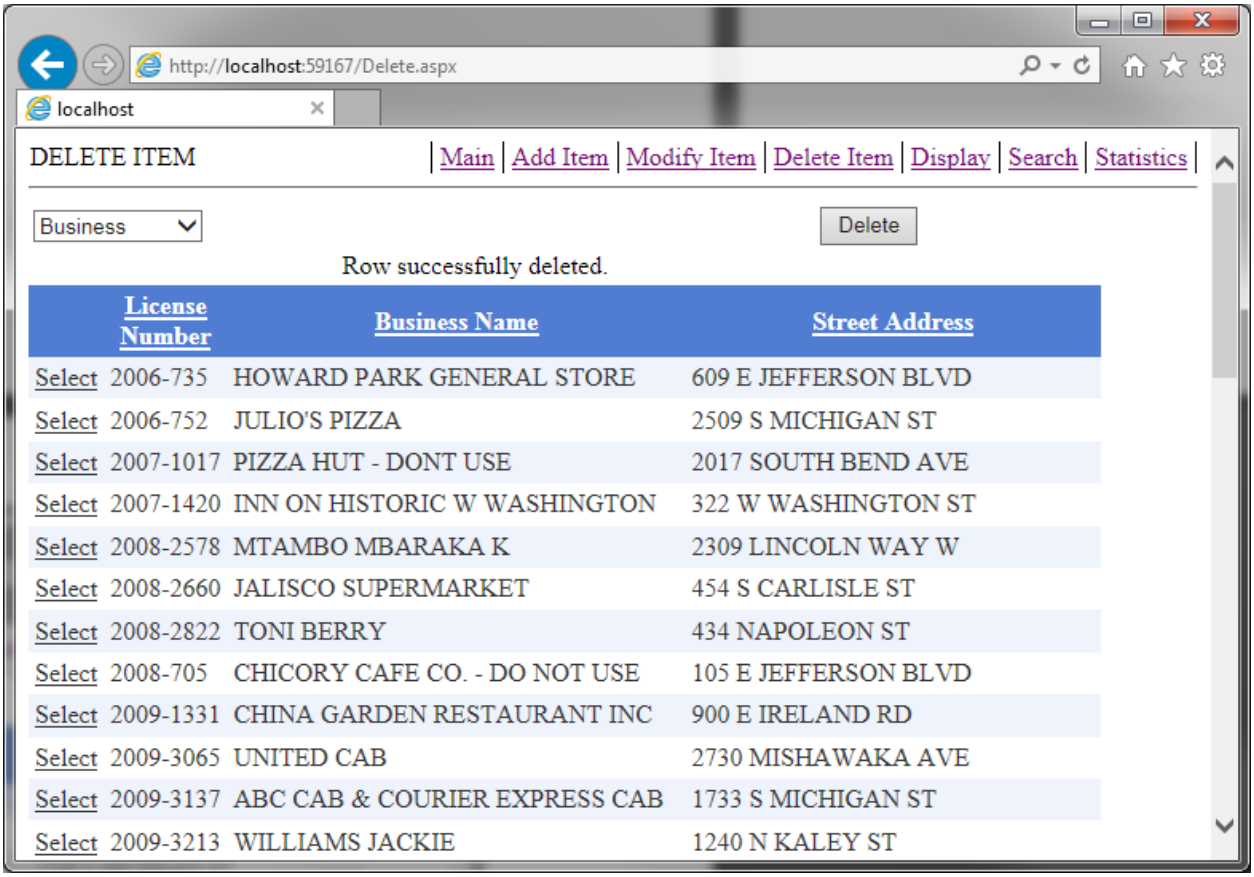


Total # of Parks:	50
Total # of Parks by Park Types:	Block Park: 15 Cemetery: 2 Community Park: 8 Golf Course: 4 Memorial: 1 Neighborhood Park: 19 Zoo: 1
Total # of Businesses:	50
Total # of license renewals for each Business:	110 E Angela Blvd: 1 Alarms By Turner: 1 Arborcare Inc: 1 Burton's Resale: 1 Chicory Cafe Co. - Do Not Use: 1 China Garden Restaurant Inc: 1 El Paraiso: 1 Golden Dragon Chinese Rest: 1 Howard Park General Store: 1 Howard Park Ice Rink: 1 Inn On Historic W Washington: 1 Jalisco Supermarket: 1 Jp Morgan Chase Corporate Sec: 1 Julio's Pizza: 1 Let's Spoon Frozen Yogurt / Cl: 1 Lil J Bueno's Body Shop & Auto: 1 Logan's Roadhouse: 1 Main Street Coffee House Inc: 1 Meijer Gas Station #120: 1 Mibu Servicer Inc: 1 Pizza Hut - Dont Use: 1 Residence Inn: 1 St Joseph Co 4-H Fair Inc: 1 That Cheesecake Place: 1 Tom's Car Care Center Inc: 1 Torch Lounge Inc: 1
Total # of Facilities that have substring "Fire":	22

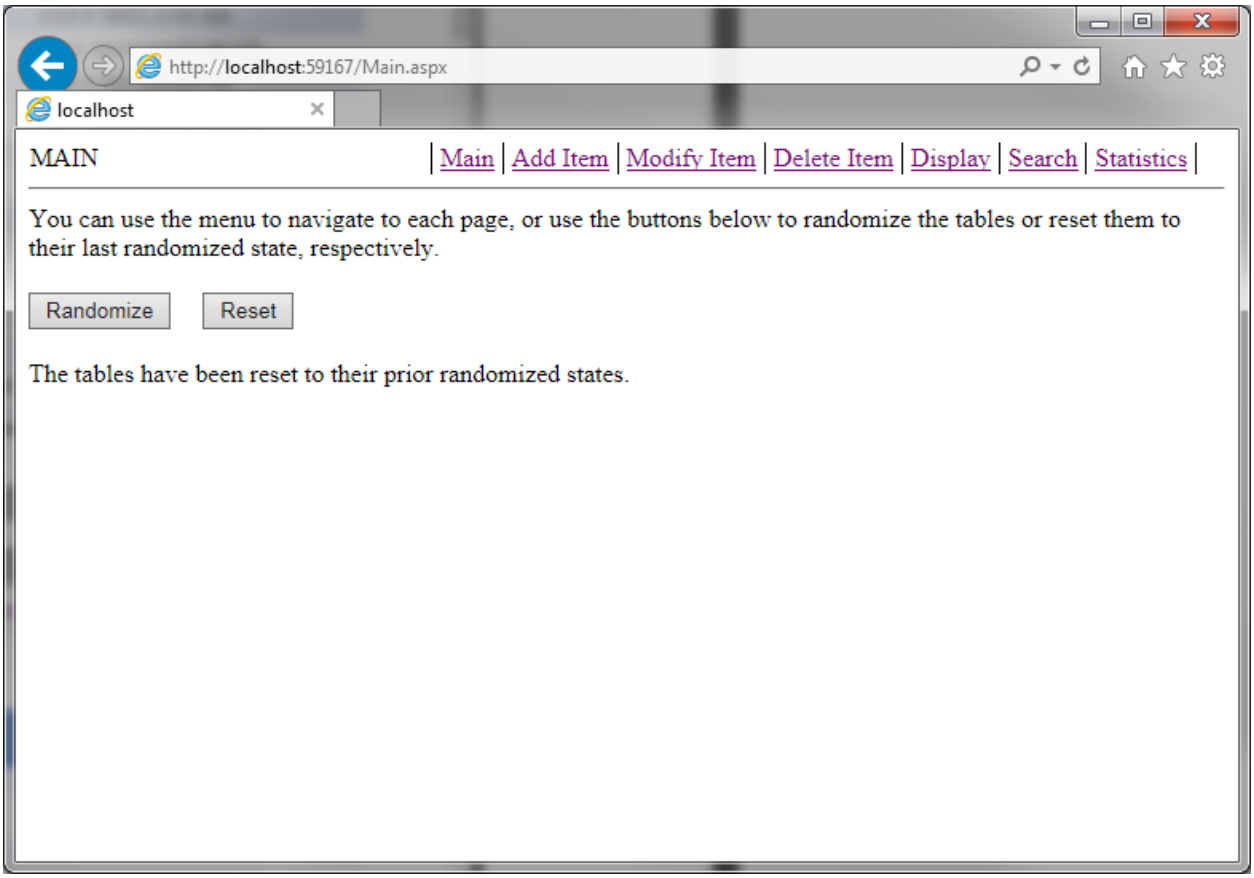
Selecting a sacrificial business for deletion to test the reset funcionality



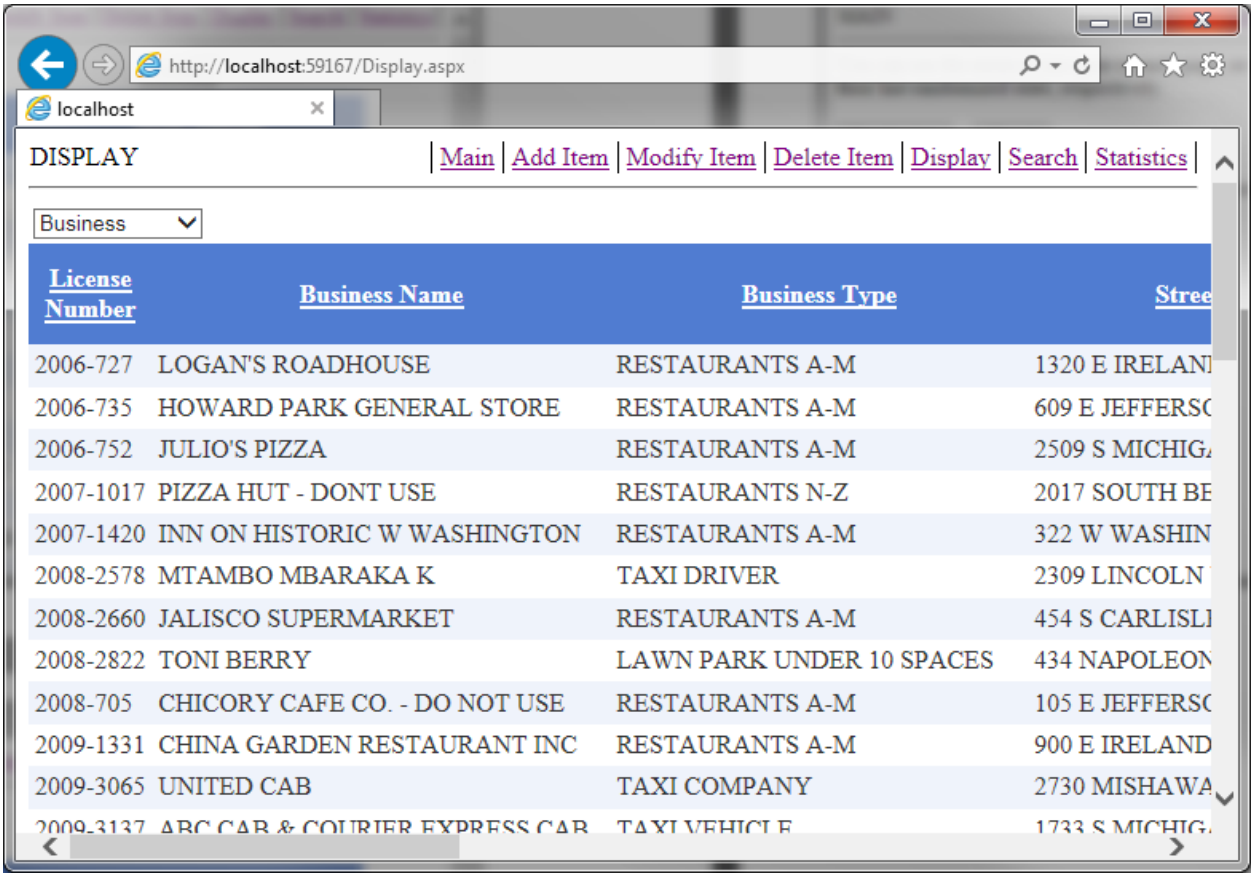
Goodbye



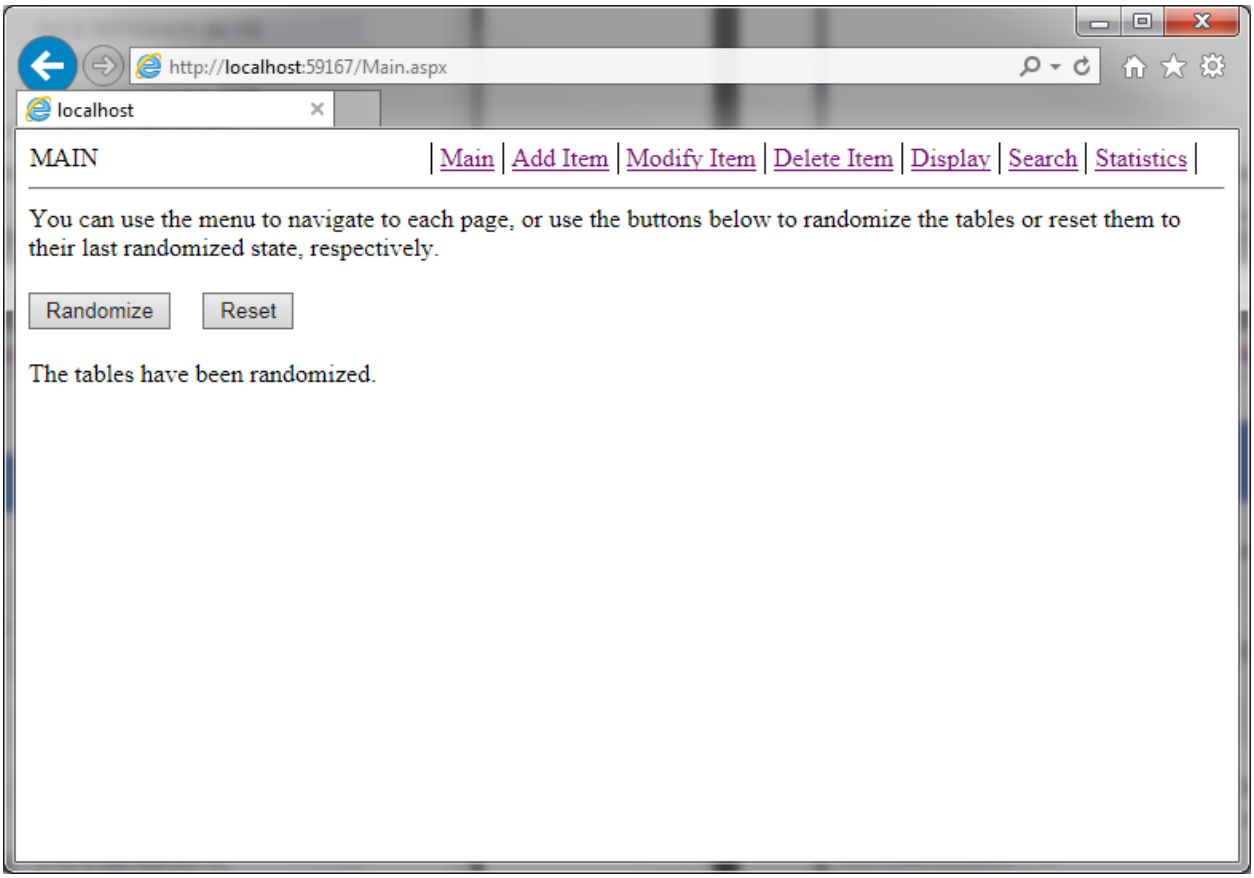
Tables have been reset



Sacrificial business has made a miraculous recovery



Randomize everything



Presto-chango, different data