2011

COOKBOOK

Access Control Rules

Table of Contents

Web app generator **Code On Time** now includes *Access Control Rules*, the first component of EASE (Enterprise Application Services Engine). EASE components simplify implementation of enterprise-class features in line-of-business web applications created with Code On Time and require little to no programming at all.

Access Control Rules

Many line-of-business applications start with a simple spreadsheet that allows performing data analysis or calculations to satisfy a specific business requirement. The spreadsheet turns in an application as soon as business users realize that the spreadsheet must accept data input from multiple users. A database is set up and an application user interface is built on top of it.

In some instances the entire database contents may be available to all authorized application end users. In most situations there is a need to create a set of restrictions that would separate slices of data that are available to individual users or users in a given business role.

For example, a web application administrator may be authorized to see the entire list of customers. A sales person will likely be allowed to see only the customers that he or she has a relationship with.

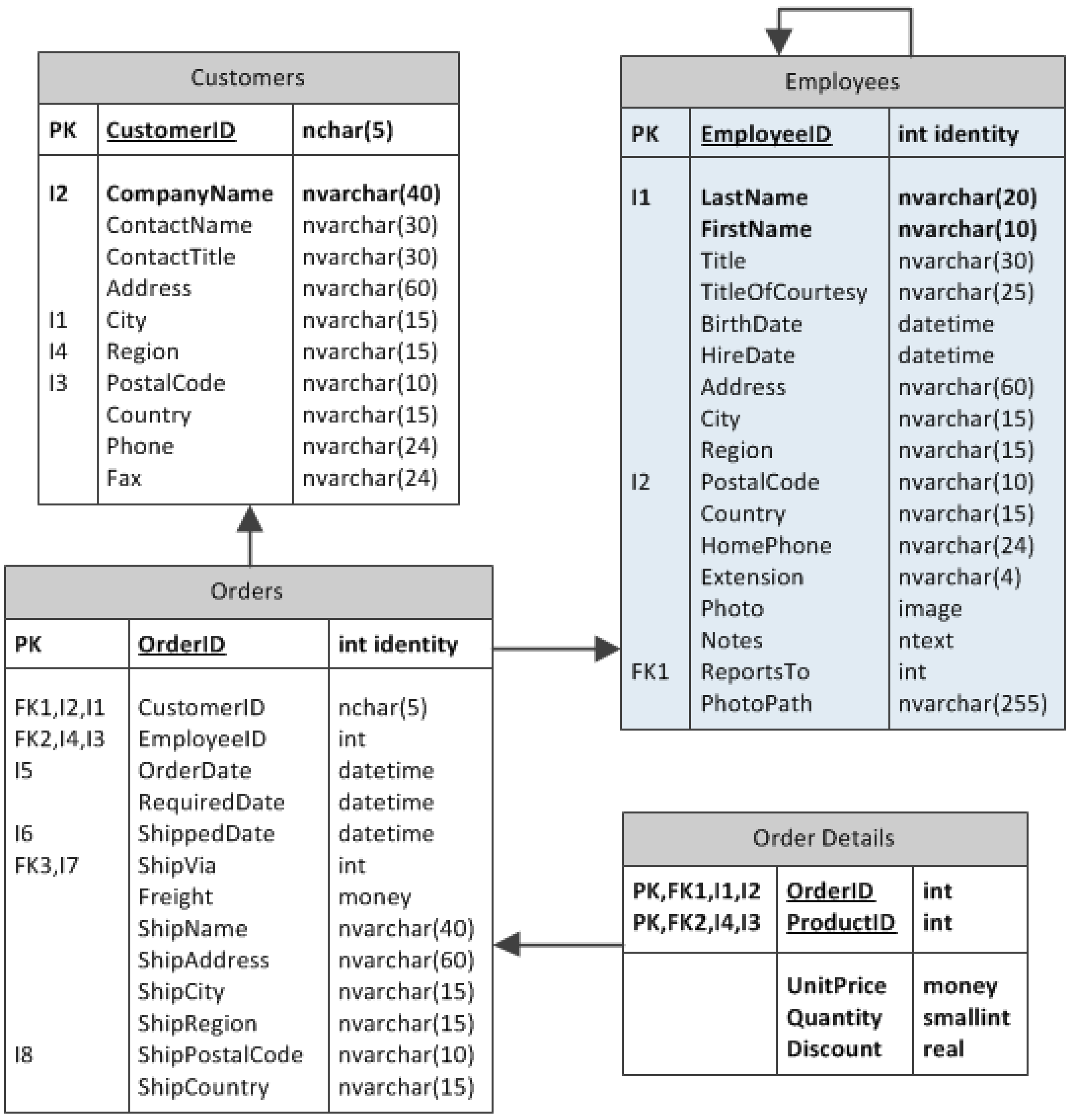
Another example is a line-of-business web application integrated with a content management system such as *DotNetNuke* or *Microsoft SharePoint*. The application data must be isolated by a portal or site name. Data created by all CMS users is stored in the same database tables but under no circumstances shall the users see each other's records.

Preparing Database for Access Control

A common approach to facilitate access control implementation is to add columns reflecting ownership of data. For example, you can implement a User ID column in each table of your database.

If more than one user has a relationship with a data record then developers opt to implement a dedicated table linking a data record with multiple users.

A typical relational database structure provides certain natural means of establishing access control based on database table relationships. It may be sufficient to implement just one ownership column. For example, consider the following snippet from the *Northwind* sample.



Customers

PK	CustomerID	nchar(5)
I2	CompanyName	nvarchar(40)
	ContactName	nvarchar(30)
	ContactTitle	nvarchar(30)
	Address	nvarchar(60)
I1	City	nvarchar(15)
I4	Region	nvarchar(15)
I3	PostalCode	nvarchar(10)
	Country	nvarchar(15)
	Phone	nvarchar(24)
	Fax	nvarchar(24)

Employees

PK	EmployeeID	int identity
I1	LastName	nvarchar(20)
	FirstName	nvarchar(10)
	Title	nvarchar(30)
	TitleOfCourtesy	nvarchar(25)
	BirthDate	datetime
	HireDate	datetime
	Address	nvarchar(60)
	City	nvarchar(15)
	Region	nvarchar(15)
I2	PostalCode	nvarchar(10)
	Country	nvarchar(15)
	HomePhone	nvarchar(24)
	Extension	nvarchar(4)
	Photo	image
	Notes	ntext
FK1	ReportsTo	int
	PhotoPath	nvarchar(255)

Orders

PK	OrderID	int identity
FK1,I2,I1	CustomerID	nchar(5)
FK2,I4,I3	EmployeeID	int
I5	OrderDate	datetime
	RequiredDate	datetime
I6	ShippedDate	datetime
FK3,I7	ShipVia	int
	Freight	money
	ShipName	nvarchar(40)
	ShipAddress	nvarchar(60)
	ShipCity	nvarchar(15)
	ShipRegion	nvarchar(15)
I8	ShipPostalCode	nvarchar(10)
	ShipCountry	nvarchar(15)

Order Details

PK,FK1,I1,I2	OrderID	int
PK,FK2,I4,I3	ProductID	int
	UnitPrice	money
	Quantity	smallint
	Discount	real

Table *Orders* has a reference to an *Employee*. If a web application user is in the role of *Sales* then one can make the following assumptions:

•an employee shall see only his or her orders

•an employee shall see only customers that have matching orders placed by the employee

•a subset of order details is naturally visible to an employee when an order is selected

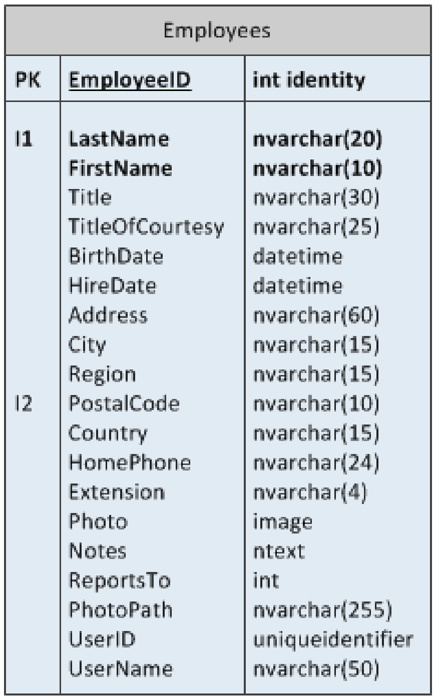
•if an employee is viewing the global list of order details then only orders of her customers shall be accessible.

The described access control rules can be implemented if we associate each *Employees* table record with a *User ID*.

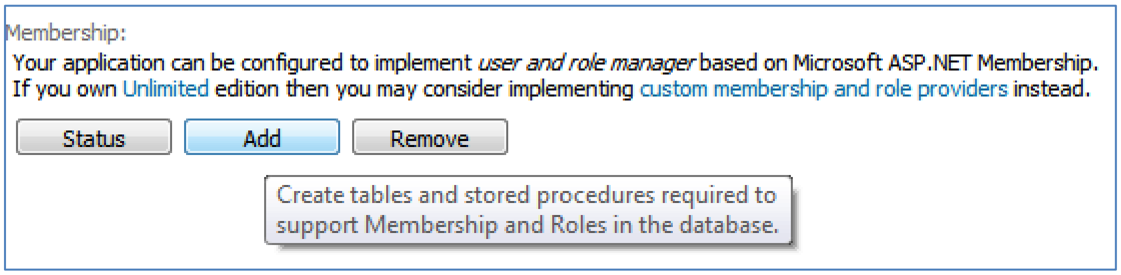
Code On Time allows integrating *Microsoft ASP.NET Membership* in the generated applications. User accounts and roles are stored in dedicated tables and user and role management APIs are already built in ASP.NET. Let's incorporate *ASP.NET Membership* plumbing in the application database.

Change *Employees* table to implement two additional columns *UserID* and *UserName*. Notice that if you are developing with a database other than *Microsoft SQL Server* then you may need to choose different types for *UserID* and *UserName* columns. For example, *MySQL* implementation of *ASP.NET Membership* uses *int* as the type of the membership user ID.

Start **Code On Time** and create a new *Web Site Factory* project. On the *Database Connection* page of project wizard, press the button next to connection string input field.



Employees		
PK	EmployeeID	int identity
I1	**LastName**	**nvarchar(20)**
	FirstName	**nvarchar(10)**
	Title	nvarchar(30)
	TitleOfCourtesy	nvarchar(25)
	BirthDate	datetime
	HireDate	datetime
	Address	nvarchar(60)
	City	nvarchar(15)
	Region	nvarchar(15)
I2	PostalCode	nvarchar(10)
	Country	nvarchar(15)
	HomePhone	nvarchar(24)
	Extension	nvarchar(4)
	Photo	image
	Notes	ntext
	ReportsTo	int
	PhotoPath	nvarchar(255)
	UserID	uniqueidentifier
	UserName	nvarchar(50)



Membership:
Your application can be configured to implement *user and role manager* based on Microsoft ASP.NET Membership.
If you own Unlimited edition then you may consider implementing custom membership and role providers instead.

Create tables and stored procedures required to support Membership and Roles in the database.

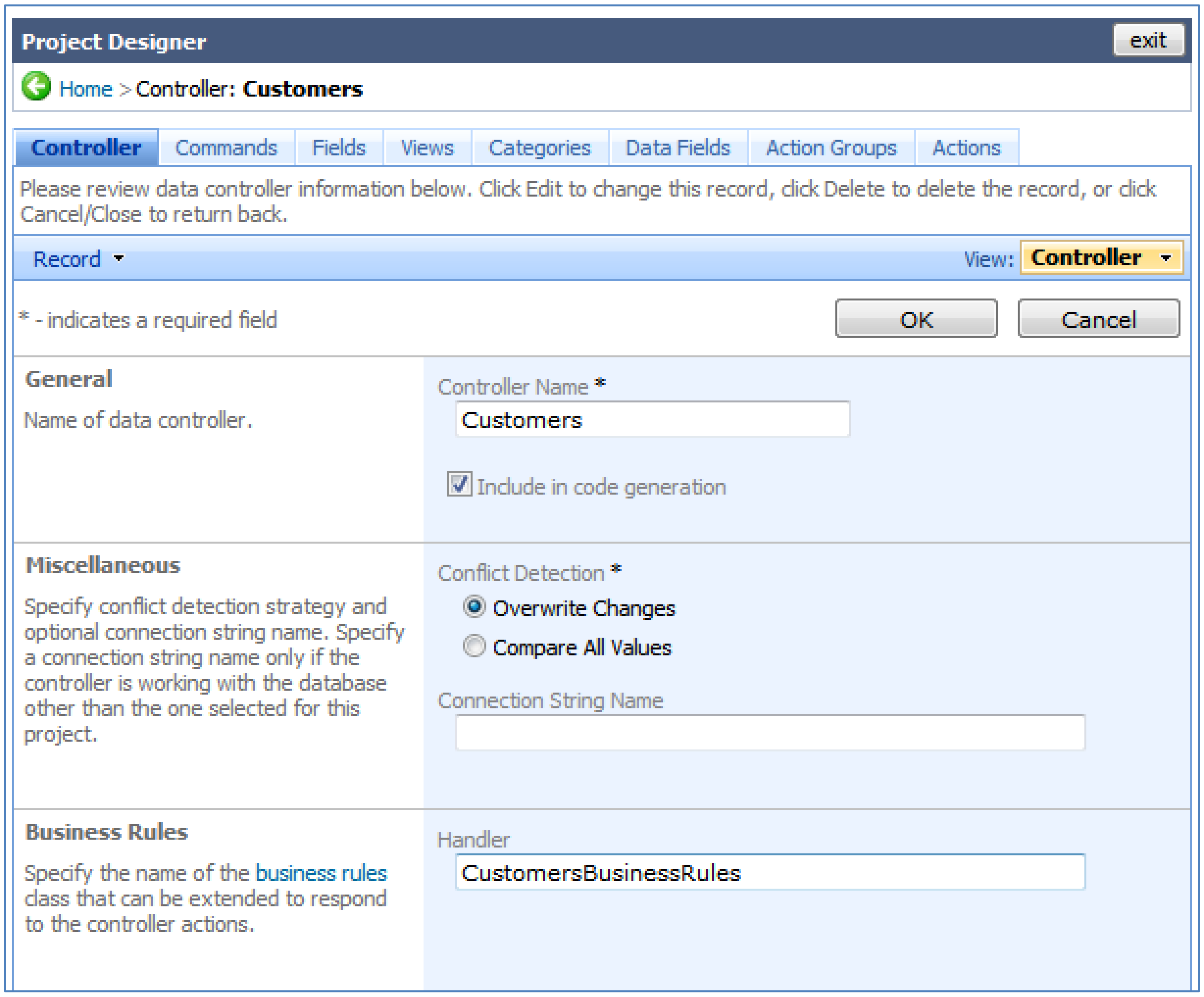
Set the database connection string and press the *Add* button to create membership tables and stored procedures in the application database.

Creating a Business Rules Class

Access Control Rules in **Code On Time** web applications are implemented as methods in business rules classes. These methods can be shared by all data controllers of your application or designed to address the needs of a specific data controller. We will consider both situations and will show the real-live examples of custom and shared access control rules.

We will start by creating a business rules class associated with *Customers* data controller. Click on the *Start Designer* button and on the All Controllers tab, select *Customers* data controller. Under the

Business Rules section, enter “CustomersBusinessRules” in the *Handler* field. Press *OK* to persist changes.



Please review data controller information below. Click Edit to change this record, click Delete to delete the record, or click Cancel/Close to return back.

* - indicates a required field

General

Name of data controller.

Controller Name *: Customers

[✓] Include in code generation

Miscellaneous

Specify conflict detection strategy and optional connection string name. Specify a connection string name only if the controller is working with the database other than the one selected for this project.

Conflict Detection *: [● Overwrite Changes] ○ Compare All Values

Connection String Name:

Business Rules

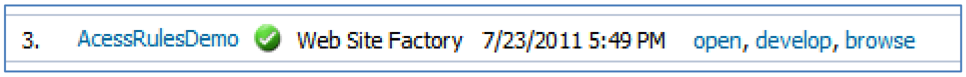
Specify the name of the business rules class that can be extended to respond to the controller actions.

Handler: CustomersBusinessRules

Click the *Exit* button at the top of the screen and press *Next* to generate the project.

The implementation of the business rule will require source code text editing and can be done in *Notepad* or a development tool such as *Visual Studio* or *Visual Web Developer*.

If you do not have a development tool on your computer then click “open” under Actions column of the Code On Time start page.



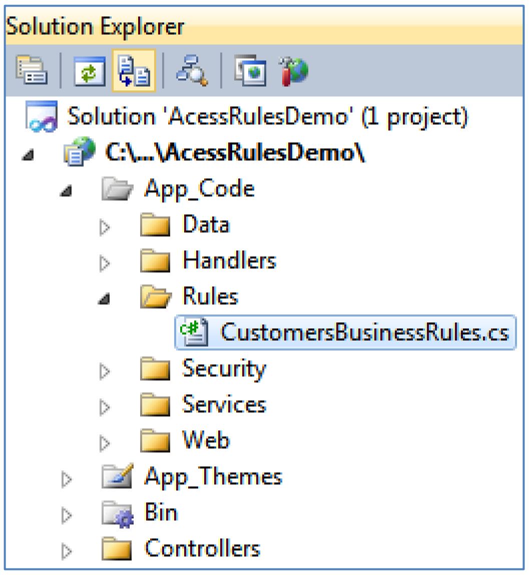
3.	AcessRulesDemo ✓	Web Site Factory	7/23/2011 5:49 PM	open, develop, browse

Select *App_Code\Rules\CustomersBusinessRules.cs(vb)* file in *Windows Explorer* and open the file in *Notepad*.

If you do have a development tool listed above then simply click “develop” link to activate the development tool and open the same file in *Solution Explorer*.

The extension of the file depends on your programming language. Double click the file name to open the file in the editor.

Next we will show examples in both *C#* and *Visual Basic* demonstrating various way of implementing access control rules.



Challenges in Implementing Access Control Rules

Modern database software makes it easy to select data. The application developer can write a data selection statement in declarative language SQL. Such statements typically list the source tables, table columns, and table join instructions. Data selection statements are frequently enhanced with filters to present the data that users would like to see.

Then there is this pesky requirement to segment the data based on user identity or business role. A developer will have to incorporate access control filters in every single data selection statement and foresee all sorts of business requirements that may call for exceptions to some of the filters.

Because of that developers end up writing their data selection statement as stored procedures persisted in the database. These stored procedures are fundamentally basic selection statements enhanced with numerous checks and conditions to ensure proper access control.

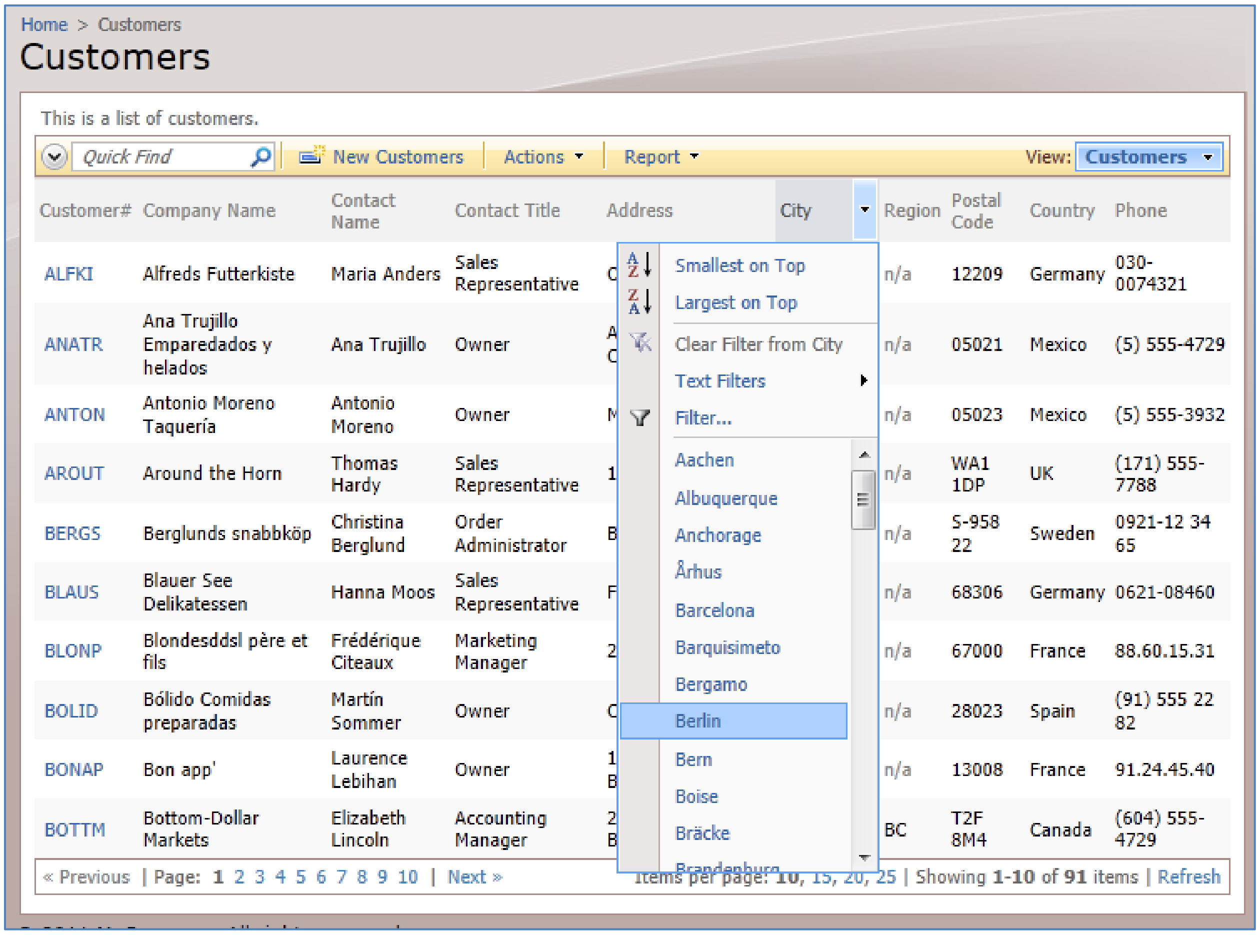
Code On Time applications create SQL selection statements on-the-fly which incorporate user-defined adaptive filters, search criteria, and sort order. Selection statements are enhanced with parameters to prevent any possibility of an injection attack that plagues many applications with hand-written SQL.

It is impossible to write a stored procedure that will accept an unknown number of filtering and sorting parameters to match on-the-fly SQL statements created by your **Code On Time** web application. *Access Control Rules* are designed specifically to address the need to apply access control restrictions to dynamic SQL statements.

Next we will show you several examples of *Access Control Rules* and will implement filtering based on user identity and business role.

Restricting Access by a Single Value

Consider the list of customers presented in the following screenshot. We can see customers from many different countries.



Home > Customers

Customers

This is a list of customers.

Customer#	Company Name	Contact Name	Contact Title	Address	City	Region	Postal Code	Country	Phone
ALFKI	Alfreds Futterkiste	Maria Anders	Sales Representative	C		n/a	12209	Germany	030-0074321
ANATR	Ana Trujillo Emparedados y helados	Ana Trujillo	Owner	A C		n/a	05021	Mexico	(5) 555-4729
ANTON	Antonio Moreno Taquería	Antonio Moreno	Owner	M		n/a	05023	Mexico	(5) 555-3932
AROUT	Around the Horn	Thomas Hardy	Sales Representative	1		n/a	WA1 1DP	UK	(171) 555-7788
BERGS	Berglunds snabbköp	Christina Berglund	Order Administrator	B		n/a	S-958 22	Sweden	0921-12 34 65
BLAUS	Blauer See Delikatessen	Hanna Moos	Sales Representative	F		n/a	68306	Germany	0621-08460
BLONP	Blondesddsl père et fils	Frédérique Citeaux	Marketing Manager	2		n/a	67000	France	88.60.15.31
BOLID	Bólido Comidas preparadas	Martín Sommer	Owner	C		n/a	28023	Spain	(91) 555 22 82
BONAP	Bon app'	Laurence Lebihan	Owner	1 B		n/a	13008	France	91.24.45.40
BOTTM	Bottom-Dollar Markets	Elizabeth Lincoln	Accounting Manager	2 B		BC	T2F 8M4	Canada	(604) 555-4729

« Previous | Page: 1 2 3 4 5 6 7 8 9 10 | Next » Items per page: 10, 15, 20, 25 | Showing 1-10 of 91 items | Refresh

Let's limit this list of customers to *USA* only and have this rule apply to all application users. Enter the following method in the *CustomersBusinessRules.cs(vb)* and save the file.

C#:

```
using System;
using System.Data;
using System.Collections.Generic;
using System.Linq;
using MyCompany.Data;

namespace MyCompany.Rules
{
    public partial class CustomersBusinessRules : MyCompany.Data.BusinessRules
    {
        [AccessControl("Customers", "Country")]
        public void CountryFilterThatAppliesToEverybody()
        {
            RestrictAccess("USA");
        }
    }
}
```

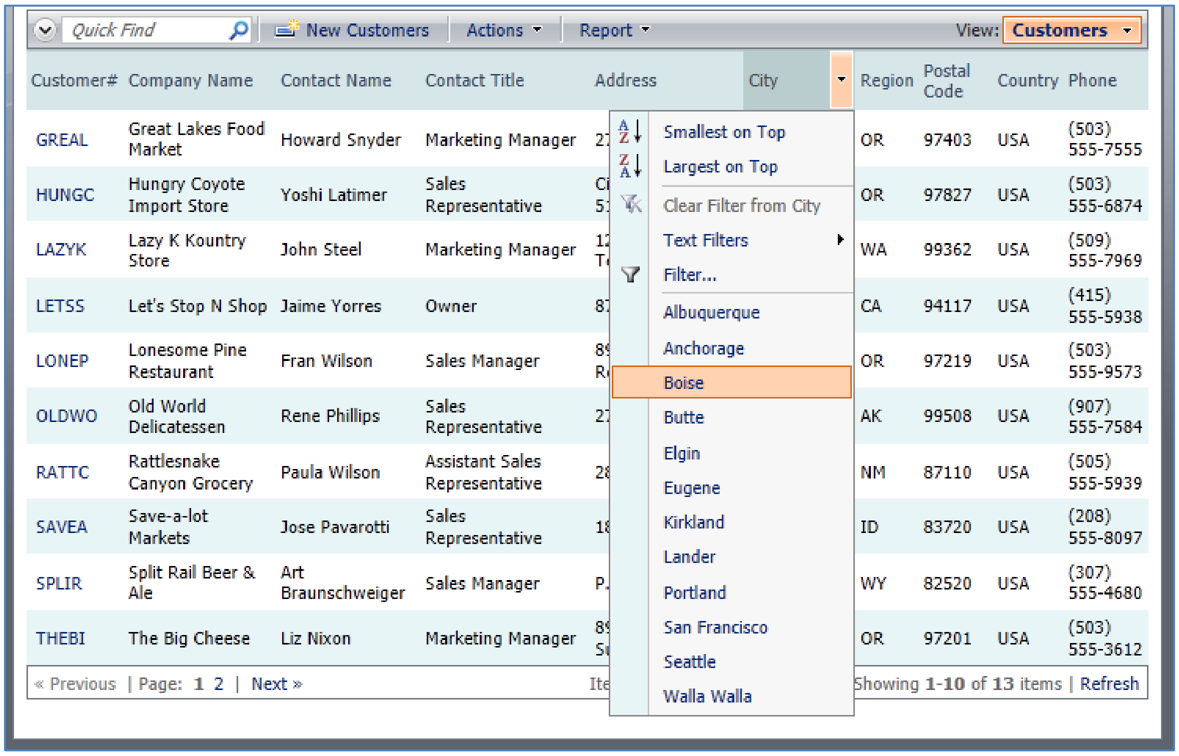
Visual Basic:

```
Imports MyCompany.Data
Imports System
Imports System.Collections.Generic
Imports System.Data
Imports System.Linq

Namespace MyCompany.Rules

    Partial Public Class CustomersBusinessRules
        Inherits MyCompany.Data.BusinessRules

        <AccessControl("Customers", "Country")> _
        Public Sub CountryFilterThatAppliesToEverybody()
            RestrictAccess("USA")
        End Sub
    End Class
End Namespace
```

The name of the method plays no role. Use your imagination to ensure that it will be easy to understand the purpose of the method in the future. The method is decorated with *AccessControl* attribute .

The first parameter of the attribute constructor indicates the data controller that must take into account this access control method. The second parameter indicates the field that will be filtered by the access control method.

If the data controller is requested to retrieve data then it will scan the list of methods of the associated business rules class and find those that are decorated with the matching *AccessControl* attribute. By default the data controller will assume that the method is designed to allow access to data. The data controller will invoke the method and if any calls to *RestrictAccess* are executed then every restriction will be incorporated in the access control filter of SQL data selection statement. In this instance the method will ensure that all users are restricted to see customers from *USA* regardless of their identity or business role.



Customer#	Company Name	Contact Name	Contact Title	Address	City	Region	Postal Code	Country	Phone
GREAL	Great Lakes Food Market	Howard Snyder	Marketing Manager	27		OR	97403	USA	(503) 555-7555
HUNGC	Hungry Coyote Import Store	Yoshi Latimer	Sales Representative	Ci 51		OR	97827	USA	(503) 555-6874
LAZYK	Lazy K Kountry Store	John Steel	Marketing Manager	12 T		WA	99362	USA	(509) 555-7969
LETSS	Let's Stop N Shop	Jaime Yorres	Owner	87		CA	94117	USA	(415) 555-5938
LONEP	Lonesome Pine Restaurant	Fran Wilson	Sales Manager	89 R		OR	97219	USA	(503) 555-9573
OLDWO	Old World Delicatessen	Rene Phillips	Sales Representative	27		AK	99508	USA	(907) 555-7584
RATTC	Rattlesnake Canyon Grocery	Paula Wilson	Assistant Sales Representative	28		NM	87110	USA	(505) 555-5939
SAVEA	Save-a-lot Markets	Jose Pavarotti	Sales Representative	18		ID	83720	USA	(208) 555-8097
SPLIR	Split Rail Beer & Ale	Art Braunschweiger	Sales Manager	P.		WY	82520	USA	(307) 555-4680
THEBI	The Big Cheese	Liz Nixon	Marketing Manager	89 S		OR	97201	USA	(503) 555-3612

« Previous | Page: 1 2 | Next » Ite Showing 1-10 of 13 items | Refresh

Notice that the list of options available in the adaptive filter is considerably shorter. Any user-defined filters will be applied on top of the filters produced by *Access Control Rules*. The access control filter in the example is activated in all instances of requests to retrieve a list of customers.

Access control methods can utilize any sort of logic to decide if a restriction is applicable at any given moment. For example, a filter can prevent access to the list of customers during non-working hours. The following access control method will allow access to the list of customers from 9 AM to 5:30 PM. If the data is being accessed outside of this time period then a restriction by a non-exiting country is activated.

C#:

```
[AccessControl("Customers", "Country")]
private void CountryFilterThatAppliesToEverybody()
{
    DateTime today = DateTime.Now;
    DateTime workDayBegins = new DateTime(today.Year, today.Month, today.Day, 9, 00, 00);
    DateTime workDayEnds = new DateTime(today.Year, today.Month, today.Day, 17, 30, 00);
    if (workDayBegins <= today && today <= workDayEnds)
        RestrictAccess("USA");
    else
        RestrictAccess("*****");

}
```

Visual Basic:

```
<AccessControl("Customers", "Country")> _
Private Sub CountryFilterThatAppliesToEverybody()
    Dim today As DateTime = DateTime.Now
    Dim workdayBegins As DateTime = New DateTime(today.Year, today.Month, today.Day, 9, 0, 0)
    Dim workdayEnds As DateTime = New DateTime(today.Year, today.Month, today.Day, 17, 30, 0)
    If (workdayBegins <= today And today <= workdayEnds) Then
        RestrictAccess("USA")
    Else
        RestrictAccess("*****")
    End If
End Sub
```

Multiple *AccessControl* attributes can be applied to the same method. If several methods with *AccessControl* attribute are discovered then the data controller will incorporate SQL data selection statement restrictions by concentrating them with “and” logic.

Restricting Access by Multiple Values

If more than one value must be used to filter out the data then simply call *RestrictAccess* method multiple times and pass the values that are compatible with the data type of the field specified in *AccessControl* attribute.

The following method will limit the list of customers to three specific IDs when presenting the list to users with roles other than Administrators.

C#:

```
        [AccessControl("CustomerID", AccessPermission.Allow)]
        protected void NonAdministrativeUsersAreAuthorizedToSeeOnlyThreeCustomers()
        {
            if (UserIsInRole("Administrators"))
                UnrestrictedAccess();
            else
            {
                RestrictAccess("GREAL");
                RestrictAccess("OLDWO");
                RestrictAccess("THEBI");
            }
        }
```

Visual Basic:

```
<AccessControl("CustomerID", AccessPermission.Allow)> _
        Protected Sub NonAdministrativeUsersAreAuthorizedToSeeOnlyThreeCustomers()
            If UserIsInRole("Administrators") Then
                UnrestrictedAccess()
            Else
                RestrictAccess("GREAL")
                RestrictAccess("OLDWO")
                RestrictAccess("THEBI")
            End If
        End Sub
```

This example is using *UnrestrictedAccess* method to indicate that the restriction does not apply to *Administrators*. Access control methods will only result in data filtering if at least one call to *RestrictAccess* has been made. You can use *UnrestrictedAccess* method to negate the result of the access restriction previously applied within the same method execution path.

Here is the slightly shorter version of the method restricting access to specific customer accounts for non-administrative users. If a user does not belong to role *Administrators* then no restrictions will be imposed on the list of customers.

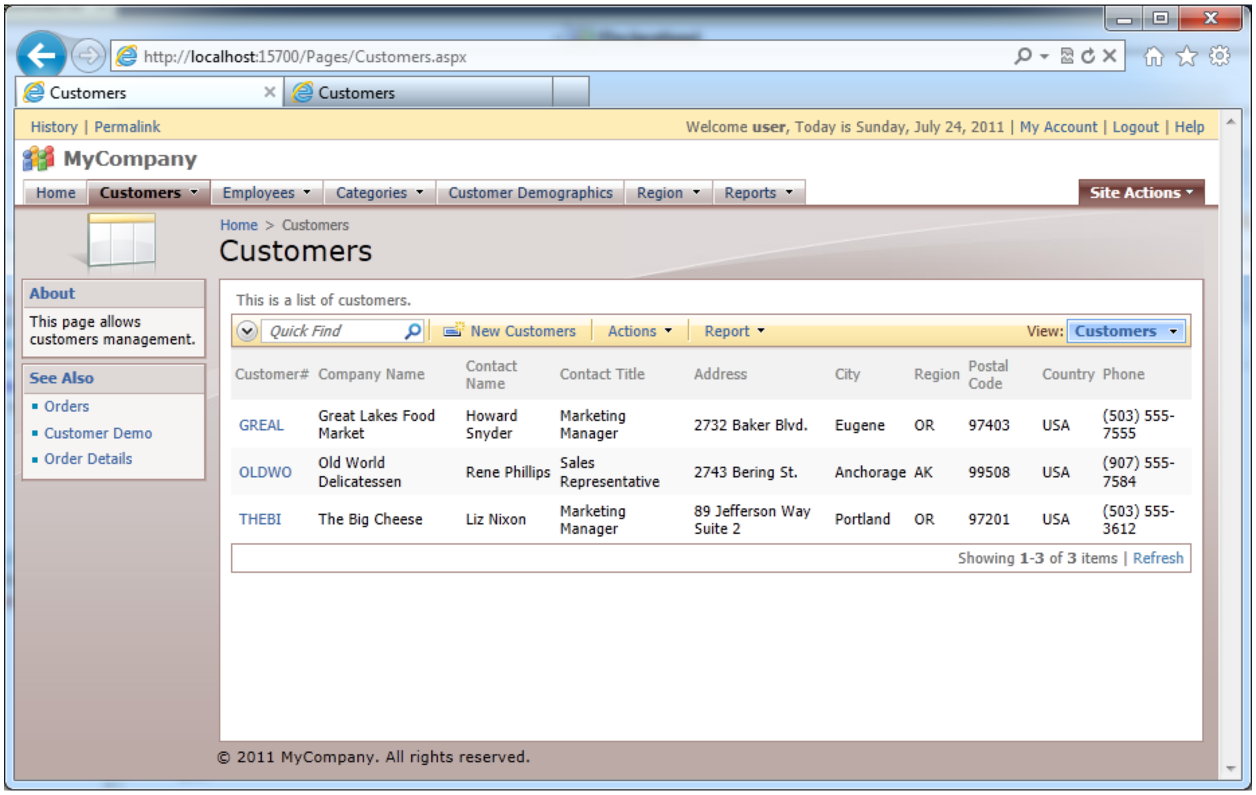
C#:

```
    [AccessControl("CustomerID", AccessPermission.Allow)]
    protected void NonAdministrativeUsersAreAuthorizedToSeeOnlyThreeCustomers()
    {
        if (!UserIsInRole("Administrators"))
        {
            RestrictAccess("GREAL");
            RestrictAccess("OLDWO");
            RestrictAccess("THEBI");
        }
    }
```

Visual Basic:

```
<AccessControl("CustomerID", AccessPermission.Allow)> _
        Protected Sub NonAdministrativeUsersAreAuthorizedToSeeOnlyThreeCustomers()
            If Not UserIsInRole("Administrators") Then
                RestrictAccess("GREAL")
                RestrictAccess("OLDWO")
                RestrictAccess("THEBI")
            End If
        End Sub
```

The screenshot shows the list of customers presented to a user with role *Users*.



Customers

This page allows customers management.

This is a list of customers.

Customer#	Company Name	Contact Name	Contact Title	Address	City	Region	Postal Code	Country	Phone
GREAL	Great Lakes Food Market	Howard Snyder	Marketing Manager	2732 Baker Blvd.	Eugene	OR	97403	USA	(503) 555-7555
OLDWO	Old World Delicatessen	Rene Phillips	Sales Representative	2743 Bering St.	Anchorage	AK	99508	USA	(907) 555-7584
THEBI	The Big Cheese	Liz Nixon	Marketing Manager	89 Jefferson Way Suite 2	Portland	OR	97201	USA	(503) 555-3612

© 2011 MyCompany. All rights reserved.

Denying Access

The previous examples of access control rules are specifying *AccessPermission.Allow* value passed as a parameter of *AccessControl* attribute. This is default permission of access control rules and can be omitted.

There is always a possibility that a data access exception must be implemented. For example, you may need to prevent administrators from being able to access *The Big Cheese* customer from *USA*. The following method does just that.

C#:

```
        [AccessControl("CustomerID", AccessPermission.Deny)]
        public void ExceptionForAdministrators()
        {
            if (!UserIsInRole("Administrators"))
                UnrestrictedAccess();
            else
                RestrictAccess("THEBI");
        }
```

Visual Basic:

```
        <AccessControl("CustomerID", AccessPermission.Deny)> _
        Public Sub ExceptionForAdministrators()
            If (Not UserIsInRole("Administrators")) Then
                UnrestrictedAccess()
            Else
                RestrictAccess("THEBI")
            End If
        End Sub
```

If both “allow” and “deny” access control rules are imposing restrictions at runtime then the data controller will compose an access control filter that may looks as the one below.

(List of “Allow” restrictions**) and Not (**List of “Deny” restrictions**)**

Restricting Access via Parameterized SQL

Our examples are using static values to compose restrictions. In real world situations you may need to examine a user identity and figure the appropriate restriction value on the fly. It may also be impractical to invoke *RestrictAccess* method for each restriction due to a very large number of such restrictions. You can solve both problems by incorporating SQL statements in the definition of *AccessControl* attribute.

For example, you may want to consider restricting non-administrative accounts to see only those customers that they have a relationship with. The following method makes an assumption that the user with *User Name* = ‘user’ is a sales person and shall see only customers matching *Employee ID* = 1.

The total number of records in the *Customers* table of *Northwind* database is 91. If you implement the method presented on the next page, then the user with the name “user” will only see 65 records. Make sure to comment out or delete any previously defined access control methods described above to prevent the cumulative effect of access control restrictions.

C#:

```
        [AccessControl("CustomerID", Sql = "select distinct CustomerID from Orders where EmployeeID
= @EmployeeID")]
        public void LimitUserToSeeOnlyHerCustomers()
        {
            if (!UserIsInRole("Administrators"))
                if (Context.User.Identity.Name == "user")
                    RestrictAccess("@EmployeeID", 1);
                else
                    UnrestrictedAccess();
        }
```

Visual Basic:

```
        <AccessControl("Customers", "CustomerID", _
            "select distinct CustomerID from Orders where EmployeeID = @EmployeeID", _
            AccessPermission.Allow)> _
        Public Sub LimitUserToSeeOnlyHerCustomers()
            If (Not UserIsInRole("Administrators")) Then
                If Context.User.Identity.Name = "user" Then
                    RestrictAccess("@EmployeeID", 1)
                Else
                    UnrestrictedAccess()
                End If
            End If
        End Sub
```

The data controller will inject the SQL statement defined in *AccessControl* attribute in the data selection statement. The value of parameter *@EmployeeID* is assigned by the code in the access control method.

If you call method *RestrictAccess* with two arguments, then the first argument is the name of the parameter in SQL statement of the attribute and the second argument is its value. The data controller will compose a restriction that looks as follows.

"Customers.CustomerID" **in (***select distinct* CustomerID *from* Orders *where* EmployeeID = @EmployeeID**)**

The expression in *AccessControl* attribute is inserted as-is. It is up to you to ensure that the expression is valid and will return a correct list of values.

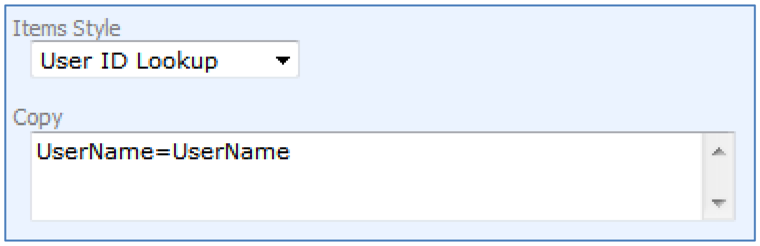
Implementing Restrictions for Northwind Sample

Let's implement access control rules in the *Northwind* sample, relying on the new columns *User ID* and *User Name* implemented at the top of this article. We will make a few changes to the web application design.

Run *Code On Time* web application generator and select your project. Click the *Next* button twice to reach the *Business Logic Layer* page in the project wizard. Select "Generate a shared business rules class …" check box and continue pressing *Next* until your reach the summary of Data Controllers in your project.

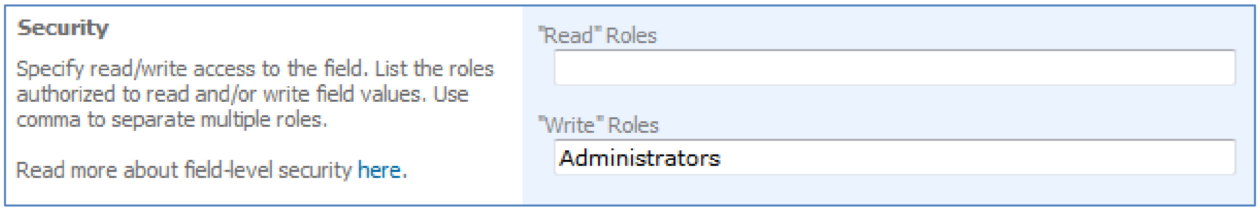
Click *Start Designer* button to activate the project designer. Select *Employees* data controller and activate *Fields* tab. Select *UserID* field, activate *Field* tab, and change its *Items Style* to "User ID Lookup".

Enter “UserName=UserName” in the *Copy* field. This instruction will ensure that the value of *UserName* field from *Membership Manager* will be copied into *Employees.UserName* field.


Items Style
User ID Lookup

Copy
UserName=UserName

Scroll down and modify the *Security* section to ensure that only *Administrators* are able to assign/create users. The following configuration will put no restrictions on who can see the user name and will enable business users with role *Administrators* to change the user account associated with the employee.


Security

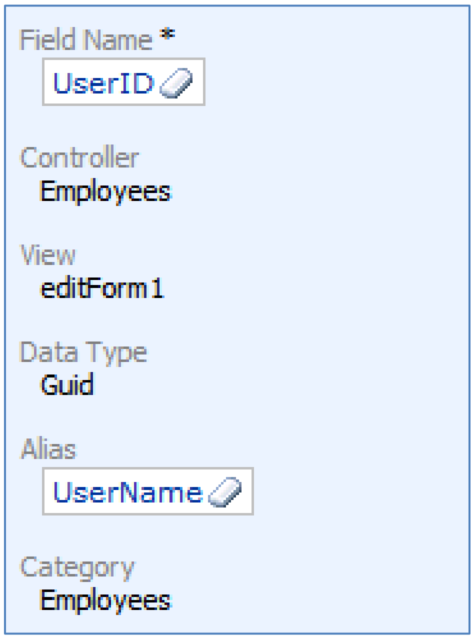
Specify read/write access to the field. List the roles authorized to read and/or write field values. Use comma to separate multiple roles.

Read more about field-level security here.

"Read" Roles

"Write" Roles
Administrators

Click *OK* to save your changes, and select this field in the list one more time. Activate *Data Fields* tab. Change both instances of the *UserID* data field to reference *UserName* field as an alias of *UserID*. Select each binding (data field) and edit *Alias* property under *General* section.


Field Name *
UserID

Controller
Employees

View
editForm1

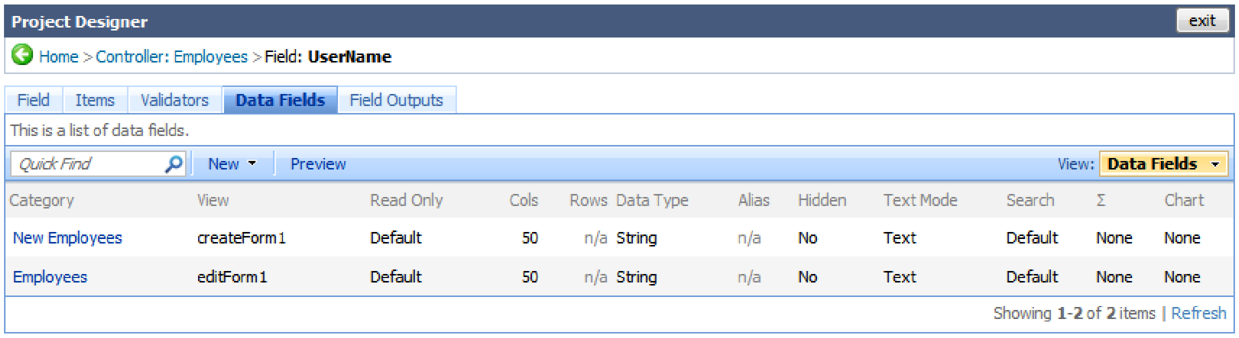
Data Type
Guid

Alias
UserName

Category
Employees

Bind *UserID* field to *grid1* view to see the *User Name* associated with each record in *Employees* table when viewing the list of employees. Click *New | New Data Field* and proceed to create a data field (binding) linking field *UserID* to view *grid1*. Make sure to leave the field *Category* blank. Choose *UserName* as the field’s *Alias*. Press *OK* to save.

Your list of data fields associated with field *UserID* will look as follows.



Project Designer exit

Home > Controller: Employees > Field: **UserName**

Field | Items | Validators | **Data Fields** | Field Outputs

This is a list of data fields.

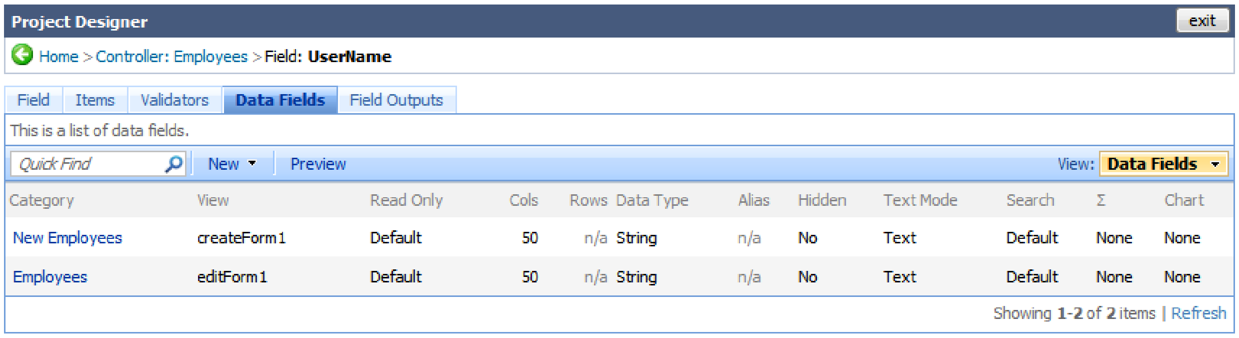
Quick Find | New | Preview | View: **Data Fields**

Category	View	Read Only	Cols	Rows	Data Type	Alias	Hidden	Text Mode	Search	Σ	Chart
New Employees	createForm1	Default	50	n/a	String	n/a	No	Text	Default	None	None
Employees	editForm1	Default	50	n/a	String	n/a	No	Text	Default	None	None

Showing **1-2** of **2** items | Refresh

Select the *Controller:Employees* link in the path at the top and select *UserName* field on the *Fields* tab.

Delete both bindings of *UserName* field to views *editForm1* and *createForm1*. These bindings are shown prior to deletion in the next screenshot.



Project Designer exit

Home > Controller: Employees > Field: **UserName**

Field | Items | Validators | **Data Fields** | Field Outputs

This is a list of data fields.

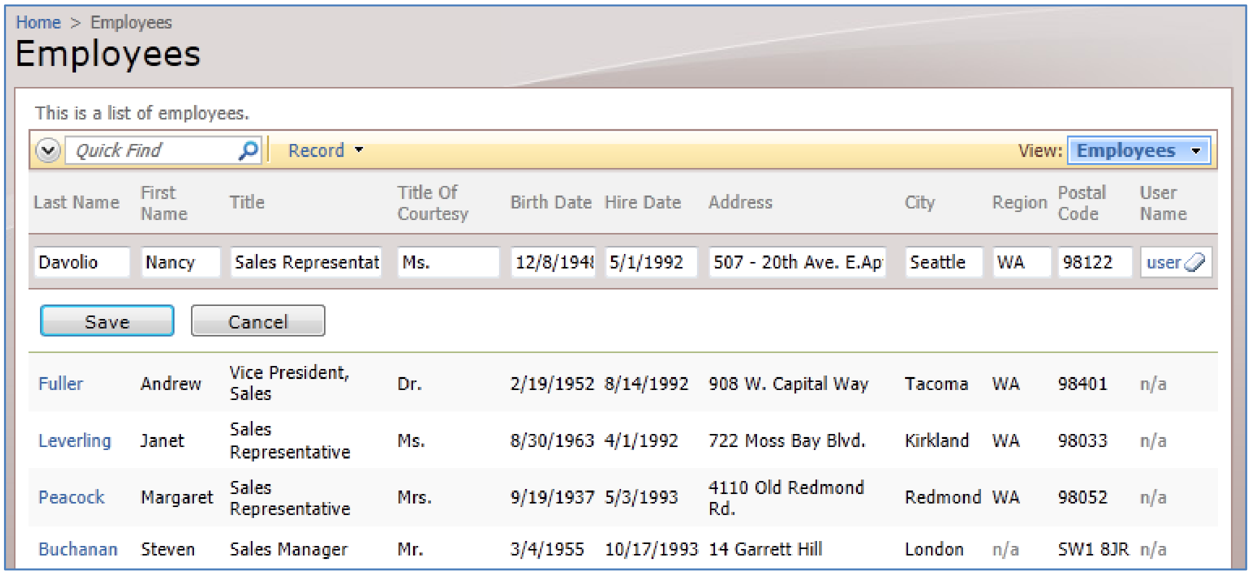
Quick Find | New | Preview | View: **Data Fields**

Category	View	Read Only	Cols	Rows	Data Type	Alias	Hidden	Text Mode	Search	Σ	Chart
New Employees	createForm1	Default	50	n/a	String	n/a	No	Text	Default	None	None
Employees	editForm1	Default	50	n/a	String	n/a	No	Text	Default	None	None

Showing **1-2** of **2** items | Refresh

Exit the *Project Designer* and generate the project.

Sign in as *admin/admin123%* and associate the user account "user" with the employee with the last name of *Davolio*.



Home > Employees

Employees

This is a list of employees.

Quick Find | Record | View: Employees

Last Name	First Name	Title	Title Of Courtesy	Birth Date	Hire Date	Address	City	Region	Postal Code	User Name
Davolio	Nancy	Sales Representat	Ms.	12/8/1948	5/1/1992	507 - 20th Ave. E.Ap	Seattle	WA	98122	user
Save	Cancel									
Fuller	Andrew	Vice President, Sales	Dr.	2/19/1952	8/14/1992	908 W. Capital Way	Tacoma	WA	98401	n/a
Leverling	Janet	Sales Representative	Ms.	8/30/1963	4/1/1992	722 Moss Bay Blvd.	Kirkland	WA	98033	n/a
Peacock	Margaret	Sales Representative	Mrs.	9/19/1937	5/3/1993	4110 Old Redmond Rd.	Redmond	WA	98052	n/a
Buchanan	Steven	Sales Manager	Mr.	3/4/1955	10/17/1993	14 Garrett Hill	London	n/a	SW1 8JR	n/a

Open the *CustomersBusinessRules.cs(vb)* file in the text editor and delete any previously defined access control rules. Also replace the base class with *MyCompany.Rules.SharedBusinessRules*. This will ensure that any global shared access control rules will be inherited in custom business rules associated with *Customers* data controller.

The new version of *CustomersBusinessRules* is presented below.

C#:

```
using System;
using System.Data;
using System.Collections.Generic;
using System.Linq;
using MyCompany.Data;

namespace MyCompany.Rules
{
    public partial class CustomersBusinessRules :
       MyCompany.Rules.SharedBusinessRules
    {
    }
}
```

Visual Basic:

```
Imports MyCompany.Data
Imports System
Imports System.Collections.Generic
Imports System.Data
Imports System.Linq

Namespace MyCompany.Rules

    Partial Public Class CustomersBusinessRules
        Inherits MyCompany.Rules.SharedBusinessRules

    End Class

End Namespace
```

Refresh the project tree in *Solution Explorer* if you are using *Visual Studio* or *Visual Web Developer*.

Open file *~/App_Code/Rules/SharedBusinessRules.cs(vb)* in the editor. This file implements *MyCompany.Rules.SharedBusinessRules* class. The data controller implementation of your web application will create an instance of this class when preparing to process requests from any data controllers of your project if a dedicated data controller is not available.

We will implement a collection of access control rules to perform consistent data selection restrictions based on user identity.

C#:

```
using System;
using System.Data;
using System.Collections.Generic;
using System.Linq;
using MyCompany.Data;

namespace MyCompany.Rules
{
    public partial class SharedBusinessRules : MyCompany.Data.BusinessRules
    {

        public object UserID
        {
            get
            {
                return System.Web.Security.Membership.GetUser().ProviderUserKey;
            }
        }

        [AccessControl("EmployeeID", Sql =
            "select EmployeeID from Employees where UserID = @UserId")]
        [AccessControl("CustomerID", Sql =
            @"select distinct CustomerID from Orders
            inner join Employees
            on Orders.EmployeeID = Employees.EmployeeID
            where Employees.UserID = @UserID")]
        public void RestrictByEmployeeIdAndCustomerId()
        {
            if (!UserIsInRole("Administrators"))
                RestrictAccess("@UserID", UserID);
        }
    }
}
```

Visual Basic:

```
Imports MyCompany.Data
Imports System
Imports System.Collections.Generic
Imports System.Data
Imports System.Linq

Namespace MyCompany.Rules

    Partial Public Class SharedBusinessRules
        Inherits MyCompany.Data.BusinessRules

        Public ReadOnly Property UserID As Object
            Get
                Return System.Web.Security.Membership.GetUser().ProviderUserKey
            End Get
        End Property

        <AccessControl(Nothing, "EmployeeID", _
            "select EmployeeID from Employees where UserID = @UserID")> _
        <AccessControl(Nothing, "CustomerID", _
            "select distinct CustomerID from Orders " & _
            "inner join Employees " & _
            "on Orders.EmployeeID = Employees.EmployeeID " & _
            "where Employees.UserID = @UserID")> _
        Public Sub RestrictByEmployeeIdAndCustomerId()
```

```
            If Not UserIsInRole("Administrators") Then
                RestrictAccess("@UserID", UserID)
            End If
        End Sub

    End Class
End Namespace
```

Paste the source code defined above in the *~/App_Code/Rules/SharedBusinessRules.cs(vb)*, save it and try it in action by signing in as *user/user123%.* You will notice that this simple restriction uniformly affects *Customers*, *Order Details*, *Employees*, *Employee Territories,* and *Customer Demographics*. Any data view that has either *CustomerID* or *EmployeeID* field will be automatically restricted if the current is not a member of *Administrators* role.

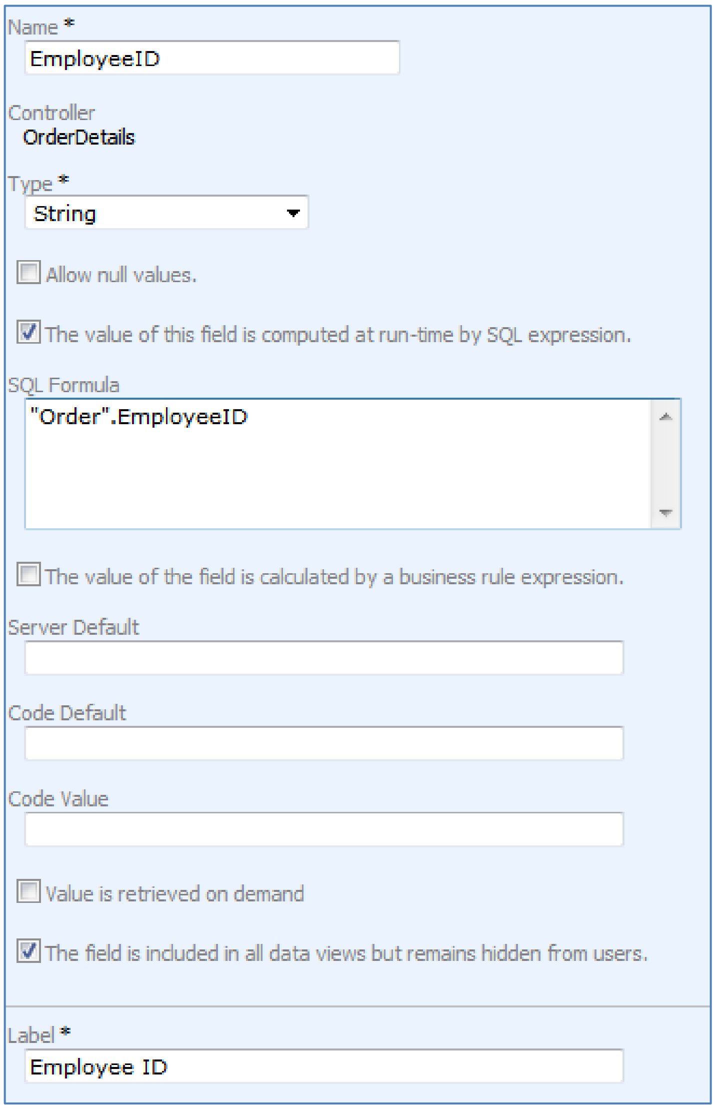
You can widen the reach of shared restrictions to *Order Details* if you simply add *EmployeeID* field to this data controller and mark it as *Hidden*.

The command text of *Order Details* data controller is defined as follows. The command is a simple SQL selection statement joining relevant tables with main table *dbo.[Order Details].*

```
select
    "OrderDetails"."OrderID" "OrderID"
    ,"Order"."CustomerID" "OrderCustomerID"
    ,"OrderCustomer"."CompanyName" "OrderCustomerCompanyName"
    ,"OrderEmployee"."LastName" "OrderEmployeeLastName"
    ,"OrderShipVia"."CompanyName" "OrderShipViaCompanyName"
    ,"OrderDetails"."ProductID" "ProductID"
    ,"Product"."ProductName" "ProductProductName"
    ,"ProductCategory"."CategoryName" "ProductCategoryCategoryName"
    ,"ProductSupplier"."CompanyName" "ProductSupplierCompanyName"
    ,"OrderDetails"."UnitPrice" "UnitPrice"
    ,"OrderDetails"."Quantity" "Quantity"
    ,"OrderDetails"."Discount" "Discount"
from "dbo"."Order Details" "OrderDetails"
    left join "dbo"."Orders" "Order" on "OrderDetails"."OrderID" =
"Order"."OrderID"
    left join "dbo"."Customers" "OrderCustomer" on "Order"."CustomerID" =
"OrderCustomer"."CustomerID"
    left join "dbo"."Employees" "OrderEmployee" on "Order"."EmployeeID" =
"OrderEmployee"."EmployeeID"
    left join "dbo"."Shippers" "OrderShipVia" on "Order"."ShipVia" =
"OrderShipVia"."ShipperID"
    left join "dbo"."Products" "Product" on "OrderDetails"."ProductID" =
"Product"."ProductID"
    left join "dbo"."Categories" "ProductCategory" on "Product"."CategoryID"
= "ProductCategory"."CategoryID"
    left join "dbo"."Suppliers" "ProductSupplier" on "Product"."SupplierID" =
"ProductSupplier"."SupplierID"
```

We will create a field that uses *Order* alias of table *dbo.Orders* to reference *Orders.EmployeeID* column in *SQL Formula*.

Start the *Project Designer* and select *Order Details* data controller. Activate *Fields* tab and choose *New | New Field* option on the action bar. Enter the following properties for the new field under *New Field* section and save its settings by pressing the *OK* button. Notice that we have wrapped the word *Order* with double quotes to make sure that it will not be misinterpreted by the database server as SQL keyword "order".



Name *
EmployeeID

Controller
OrderDetails

Type *
String

☐ Allow null values.

☑ The value of this field is computed at run-time by SQL expression.

SQL Formula
```
"Order".EmployeeID
```

☐ The value of the field is calculated by a business rule expression.

Server Default

Code Default

Code Value

☐ Value is retrieved on demand

☑ The field is included in all data views but remains hidden from users.

Label *
Employee ID

The field will be automatically available in all data views of controller *Order Details* but remain hidden from end users. The presence of the field will allow it to participate in the access control rules.

Generate the application and observe that shared business rules now extend to *Order Details* as well. Non-administrative users will only see the order details of orders that they have placed in the system.

Availability

The new access control rule mechanism greatly simplifies creation of consistent data segmentation in multi-tenant applications that require isolation of database content created by users.

Access Control Rules are available in the *Premium* and *Unlimited* editions of *Code On* Time web application generator. Owners of *Free* and *Standard* editions can use *Filter Expressions* discussed next.

What's Next?

The *Unlimited* edition of *Code On Time* generator will soon be offering *Dynamic Access Control List*, another powerful component of *EASE* (*Enterprise Application Services Engine*).

Dynamic Access Control List is designed to complement *Access Control Rules* and allow the luxury of defining precise access control at runtime. The real world business processes make it difficult to foresee all possible access control restrictions and most importantly exceptions to the rules at design time.

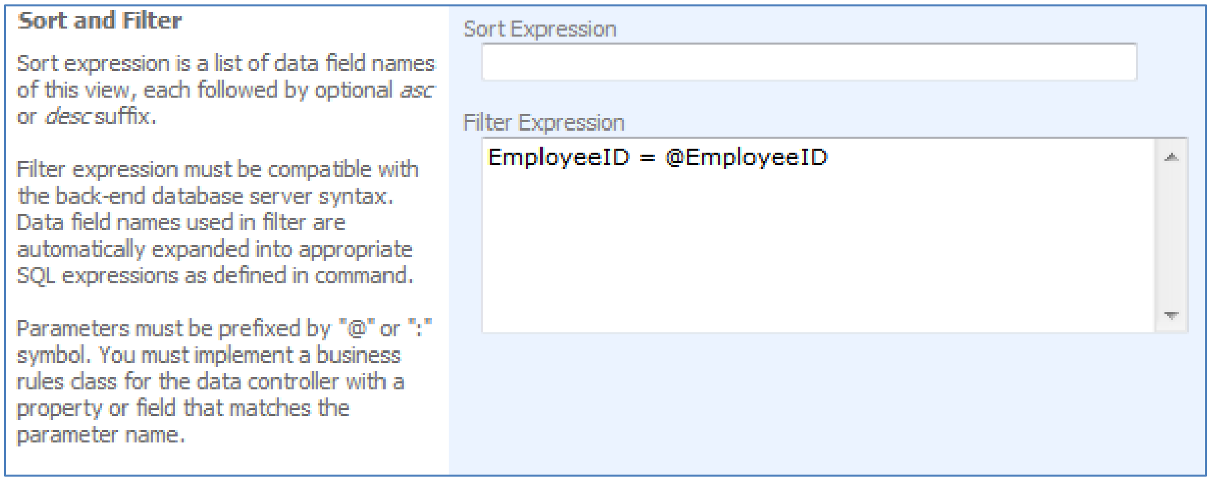
Dynamic Access Control List will maintain access control rules in the application database. The *Administrative* user interface will enable dynamic creation of access control rules to respond to business requirements at runtime without making any changes to the application code.

Filter Expressions

Developers working with *Free* and *Standard* editions can use *Filter Expressions* to implement access control rules in their applications. Filter expressions are defined on the level of a view and can reference properties of business rules classes as parameters.

For example, if you make changes to *dbo.Employees* table as described at the top of the article and add *UserID* and *UserName* columns then you can filter *Orders* by *EmployeeID*. You will have to enable shared business rules and define filter expressions on *grid1* and *editForm1* followed by the implementation of *EmployeeID* property in the *SharedBusinessRules* class.

Select your project on start page of *Code On* Time web application generator, press the *Design* button, select *Orders* data controller on *All Controllers* tab, and activate *Views* tab. Change both *grid1* and



Sort and Filter

Sort expression is a list of data field names of this view, each followed by optional *asc* or *desc* suffix.

Filter expression must be compatible with the back-end database server syntax. Data field names used in filter are automatically expanded into appropriate SQL expressions as defined in command.

Parameters must be prefixed by "@" or ":" symbol. You must implement a business rules class for the data controller with a property or field that matches the parameter name.

Sort Expression

Filter Expression

```
EmployeeID = @EmployeeID
```

editForm1 views to have the following *Filter Expression* under the section of the property page.

Generate your project and change *~/App_Code/Rules/SharedBusinessRules.cs(vb)* file to have the following code:

C#:

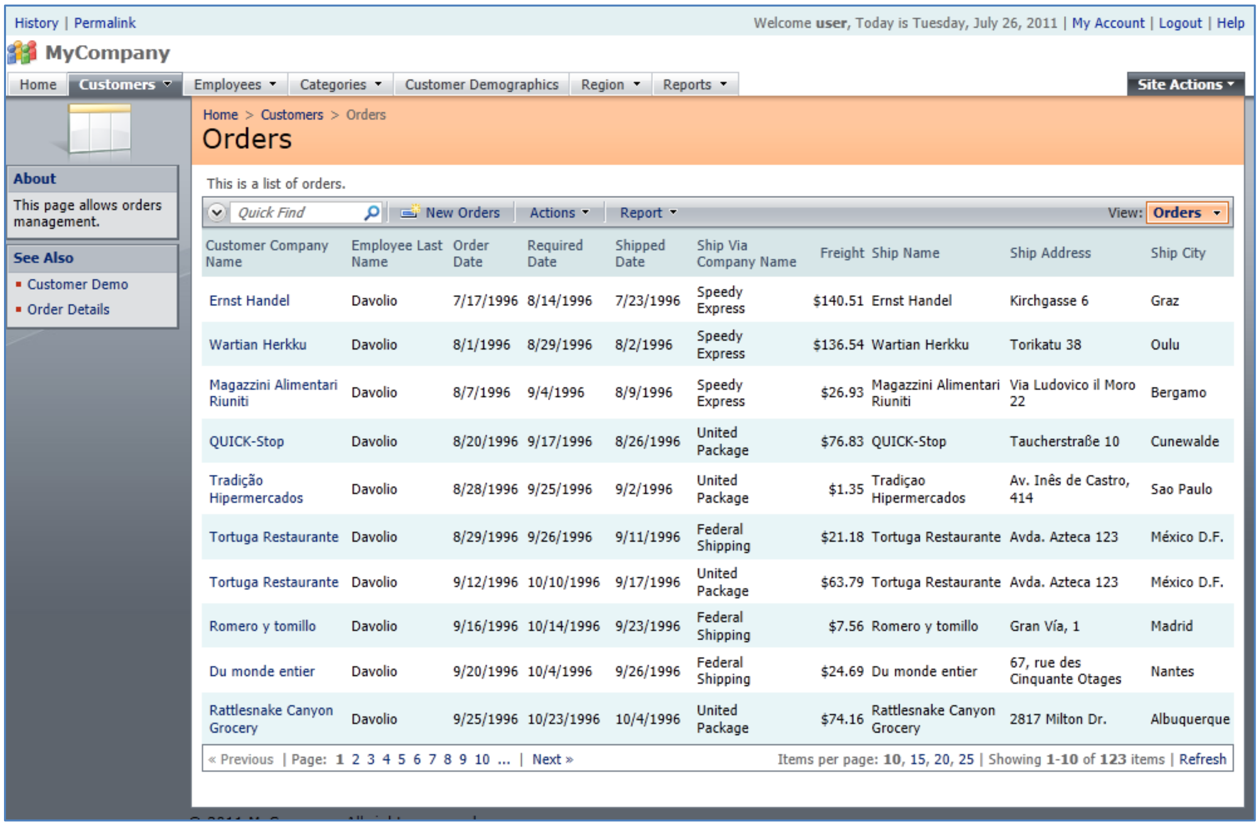
```
using System;
using System.Data;
using System.Collections.Generic;
using System.Linq;
using MyCompany.Data;
```

```
namespace MyCompany.Rules
{
    public partial class SharedBusinessRules : MyCompany.Data.BusinessRules
    {

        public object UserID
        {
            get
            {
                return System.Web.Security.Membership.GetUser().ProviderUserKey;
            }
        }

        public int EmployeeID
        {
            get
            {
                using (SqlText findEmployee = new SqlText(
                    "select EmployeeID from Employees where UserID = @UserId"))
                {
                    findEmployee.AddParameter("@UserId", UserID);
                    object id = findEmployee.ExecuteScalar();
                    if (DBNull.Value.Equals(id))
                        return -1;
                    else
                        return Convert.ToInt32(id);
                }
            }
        }

    }
}
```

Visual Basic:

```
Imports MyCompany.Data
Imports System
Imports System.Collections.Generic
Imports System.Data
Imports System.Linq

Namespace MyCompany.Rules

    Partial Public Class SharedBusinessRules
        Inherits MyCompany.Data.BusinessRules

        Public ReadOnly Property UserID As Object
            Get
                Return System.Web.Security.Membership.GetUser().ProviderUserKey
            End Get
        End Property

        Public ReadOnly Property EmployeeID As Integer
            Get
                Using findEmployee As SqlText = New SqlText( _
                    "select EmployeeID from Employees where UserID = @UserId")
                    findEmployee.AddParameter("@UserId", UserID)
                    Dim id As Object = findEmployee.ExecuteScalar()
                    If DBNull.Value.Equals(id) Then
                        Return -1
                    Else
                        Return Convert.ToInt32(id)
                    End If
                End Using
            End Get
        End Property

    End Class
End Namespace
```

The value of property *EmployeeID* will be automatically evaluated and passed as a parameter in SQL statements created to render data for presentation in *grid1* and *editForm1* views of *Orders* data controller. Here is how a user associated with employee *Davolio* may see a list of *Orders*.



Orders

This is a list of orders.

Customer Company Name	Employee Last Name	Order Date	Required Date	Shipped Date	Ship Via Company Name	Freight	Ship Name	Ship Address	Ship City
Ernst Handel	Davolio	7/17/1996	8/14/1996	7/23/1996	Speedy Express	$140.51	Ernst Handel	Kirchgasse 6	Graz
Wartian Herkku	Davolio	8/1/1996	8/29/1996	8/2/1996	Speedy Express	$136.54	Wartian Herkku	Torikatu 38	Oulu
Magazzini Alimentari Riuniti	Davolio	8/7/1996	9/4/1996	8/9/1996	Speedy Express	$26.93	Magazzini Alimentari Riuniti	Via Ludovico il Moro 22	Bergamo
QUICK-Stop	Davolio	8/20/1996	9/17/1996	8/26/1996	United Package	$76.83	QUICK-Stop	Taucherstraße 10	Cunewalde
Tradição Hipermercados	Davolio	8/28/1996	9/25/1996	9/2/1996	United Package	$1.35	Tradiçao Hipermercados	Av. Inês de Castro, 414	Sao Paulo
Tortuga Restaurante	Davolio	8/29/1996	9/26/1996	9/11/1996	Federal Shipping	$21.18	Tortuga Restaurante	Avda. Azteca 123	México D.F.
Tortuga Restaurante	Davolio	9/12/1996	10/10/1996	9/17/1996	United Package	$63.79	Tortuga Restaurante	Avda. Azteca 123	México D.F.
Romero y tomillo	Davolio	9/16/1996	10/14/1996	9/23/1996	Federal Shipping	$7.56	Romero y tomillo	Gran Vía, 1	Madrid
Du monde entier	Davolio	9/20/1996	10/4/1996	9/26/1996	Federal Shipping	$24.69	Du monde entier	67, rue des Cinquante Otages	Nantes
Rattlesnake Canyon Grocery	Davolio	9/25/1996	10/23/1996	10/4/1996	United Package	$74.16	Rattlesnake Canyon Grocery	2817 Milton Dr.	Albuquerque

« Previous | Page: 1 2 3 4 5 6 7 8 9 10 … | Next » Items per page: 10, 15, 20, 25 | Showing 1-10 of 123 items | Refresh

Class *SharedBusinessRules* is created to handle requests for all data controllers. This allows referencing *EmployeeID* as a parameter in other data controllers including *EmployeeTerritories*, *Employees*, and *Order Details*.

Filter expressions lack the flexibility of conditional restrictions available with *Access Control Rules* and *Dynamic Access Control List*. You will need to compose filter expressions that use more than one parameter to accomplish conditional filtering.

If you need to return more than one value for filtering purposes, then change the type of property to be a list of values or an array. Also refer to the property values as shown in the example below.

```
EmployeeID in @EmployeeID
```

Filter expressions can be used with *Access Control Rules* when needed if a presentation of data in a specific view requires additional filtering.