Working with View Projection

With Crosslight Entity Designer Extensions, you can generate entity services and controllers that comply with ASP.NET Web API. To learn how to create entity services, see Creating Entity Model and Services with Crosslight Entity Designer Extensions.

The generated service represents a single entity as one service. This entity service is a REST service that support OData query. To learn more about OData query, see http://www.odata.org/documentation/odata-version-2-0/uri-conventions/.

Using OData query, you can include any related tables from the targeted service using $expand query. To learn more how to include entities from entity service, see Accessing Entity Services. You can include child entities or parent entities using this approach. The only down side with this approach is that including related entities will add more data to be transferred to the client. This might slow down your application if you have large number of related entities.

To address this issue, you can use Data View to include all the information needed in a single view and then create a service that represents the Data View. This approach is valid, but in large scale application, this might be harder to manage as you will need to create new data views in the server-side every time you want to include information from related entities.

In Crosslight 4, Crosslight Entity Services provides a new feature, called view projection, allowing you to include information from your related entities into a flat object model with just simple metadata right from the client-side.
To use the view projection feature, you must use the new WebApi.v4 assembly in the server project. To make sure your projects are properly configured, please see Crosslight 4 Upgrade Guide.

On this page:

Preparing Your Model

The first step to use view projection feature is to create the value holder properties for your query. The view projection architecture is designed to be simple and intuitive to developers. Consider that you have a *Product* entity and the *Category* navigation property included in existing scenario, then in view projection, you will add a *CategoryName* property to the *Product* entity. The way view projection works is by extending existing model and gather related entities into the extended properties, making all information accessible in the flat object model instead of requiring access to navigational properties.

If you're using Crosslight Entity Designer Extensions, all the entities model are automatically generated. To extend the model, you can create new partial class as follows.

```
public partial class Product
{
    public string CategoryName { get; set; }
}

public partial class Customers
{
    public int TotalOrders { get; set; }
 public double TotalQuantity { get; set; }
 public double TotalPurchases { get; set; }
}
```

Applying View Projection

You can perform view projection to parent entity and child entities.

- Parent entity means that you would like to select a single value from the parent entity (accessing NavigationScalarProperty).
  E.g:
  Product.Category.Name
  Order.Customer.FullName
  Order.Employee.FirstName

- Child entities means that you would like to select aggregate values from child entities (accessing NavigationListProperty).
  E.g:
  Customer.Orders.Count()
  Customer.Orders.OrderDetails.Sum(Qty)
  Customer.Orders.OrderDetails.Sum(Price * Qty)

Accessing Parent Entity

To access parent entity you can use $select query with the following pattern $select={source property} as {target property} as follows.

http://localhost/data/northwind/products$select=Product/Category/Name as CategoryName

Querying Multiple View Projection

To query multiple view projection you can use "," (coma) separated value as follows.

http://localhost/data/northwind/products$select=Product/Category/Name as CategoryName, Product/Category/Description as CategoryDescription

This query will only select Product/Category/Name and Product/Category/Descriptor and stored them in designated target while the other properties are left untouched. If you want to include all properties of your entity and also the view projection, you will need to use the "*" as follows.

http://localhost/data/northwind/products$select=*, Product/Category/Name as CategoryName, Product/Category/Description as CategoryDescription

Aggregating Child Entities

To aggregate child entities you can also use $select query and use the supported aggregate formula as follows.

- Max
- Min
- Sum
- Count
- Average

http://localhost/data/northwind/customers$select=Orders/Count() as TotalOrders
http://localhost/data/northwind/customers$select=Orders/OrderDetails/Sum(Qty) as TotalQuantity

You can also using arithmetic expression in your aggregate arguments. The following are the supported arithmetic expression you can use

keyword	description
(	Open Bracket
)	Close Bracket
add	add (+) operation
sub	subtract (-) operation
mul	multiply (*) operation
div	division (/) operation

http://localhost/data/northwind/customers$select=Orders/OrderDetails/Sum(Qty mul Price) as TotalPurchases.

Using Query Descriptor

Instead of writing the OData query manually such as shown in the above examples, you can use QueryDescriptor object to define dynamic view projections through QueryDescriptor.SelectDescriptors property. See the following code example.

```
QueryDescriptor queryDescriptor1 = new QueryDescriptor();
queryDescriptor1.SelectDescriptors.Add(new SelectDescriptor("Category.Name",
"CategoryName"));

QueryDescriptor queryDescriptor2 = new QueryDescriptor();
queryDescriptor2.SelectDescriptors.Add(new SelectDescriptor("Orders",
"TotalQuantity", Aggregate.Count));
queryDescriptor2.SelectDescriptors.Add(new SelectDescriptor("Orders.OrderDetails",
"TotalPurchases", "Qty * Price"));
```

To learn more how to use QueryDescriptor, see Dynamic Data Access with Query Descriptor.

Using App Framework

If you're using Crosslight Enterprise App Framework, you can define the view projection more elegantly and easily using SelectAttribute. When accessing the entity service, the framework automatically read the SelectAttribute to determine the view projection query. The following code shows you how to use the SelectAttribute.

```
public partial class Product
{
    [NotMapped]
 [Select("Category.Name")]
 public string CategoryName { get; set; }
}

public partial class Customers
{
 [NotMapped]
 [Select("Orders", Aggregate.Count)]
    public int TotalOrders { get; set; }

 [NotMapped]
 [Select("Orders.OrderDetails", "Qty", Aggregate.Sum)]
 public double TotalQuantity { get; set; }

 [NotMapped]
 [Select("Orders.OrderDetails", "Qty * Price", Aggregate.Sum)]
 public double TotalPurchases { get; set; }
}
```

By defining the SelectAttribute, the App Framework is able to update the view projection property whenever the actual navigation property is updated. So consider that you modify the Product.CategoryID, when you save the entity, the Product.CategoryName will be updated based on the new specified CategoryID. This built-in smart feature makes data access development much easier and simple, allowing you to focus on your business process, and leave the heavy-lifting infrastructure to the framework.

Samples

To learn more about view projection and see how it works in real-world scenario, please check out the comprehensive view projection sample here: http://git.intersoftpt.com/projects/CROS-SUPP/repos/view-projection-samples/browse.

Related Topics

- Validating Data

- Creating Entity Model and Services with Crosslight Entity Designer Extensions
- Optimizing Data Access Performance
- Working With Entity Context Provider
- Tracking and Saving Changes

8 related results