Contents

A *scalar immediate execution* query is a LINQ query which performs an aggregation (such as *Count* or *Group*) or returns only one element (such as *First* or *Single*). Because these methods force immediate execution of the query they can't be directly used with asynchronous queries, but using the *AsScalarAsync* method you can **execute scalar immediate execution queries asynchronously**.

The problem

You've probably noticed something about a query like the following:

```
int ct = manager.Customers.Count();
```

It doesn't return a query object (an *EntityQuery<T>*) as other queries do. Instead, it returns the count of the items in the entity set.

Or consider another example:

```
Customer cust = manager.Customers.First();
```

It too doesn't return a query, but instead the first customer.

Both these queries are *immediate execution* queries in LINQ. They differ from the usual *deferred execution* queries which allow you to build a query in one step and execute the query at a later time. Immediate execution queries execute, well, immediately, and synchronously; you can't separate the creation of the query from the execution.

In an asynchronous environment such as Silverlight or a Windows Store application, where all queries sent to the EntityServer must be executed asynchronously, immediate execution queries pose a problem. For example, you can't do the following:

```
// Will not work!
var query = manager.Customers.First();
query.ExecuteAsync();
```

AsScalarAsync

Enter the DevForce *AsScalarAsync* operator and the *EntityScalarAsyncExtensions* . It's easiest to understand this with an example.

```
int ct = await manager.Customers.AsScalarAsync().Count();
```

This looks much the same as our earlier synchronous example, with one important difference. *AsScalarAsync* is called to convert the query to an *IEntityScalarQuery<T>* before the *Count* method is called. The query has been executed immediately, but asynchronously.

Like their synchronous counterparts, these methods can also accept a predicate. For example,

```
Employee emp = await manager.Employees.AsScalarAsync().First(e => e.LastName.StartsWith("D"));
```

You can also write more complex queries, such as the one below using an Include:

```
Employee emp = await manager.Employees.Include("Orders").AsScalarAsync().FirstOrNullEntity(e => e.Id == 1);
var orders = emp.Orders;  // Will not be pending.
```

Here's a query built dynamically:

```
var query = EntityQuery.Create(typeof(Customer));
var pd = PredicateBuilder.Make("CompanyName", FilterOperator.StartsWith, "D");
var cust = await query.Where(pd).AsScalarAsync().First();
```

The supported immediate execution methods are: *All, Any, Average, Contains, Count, First, FirstOrDefault, FirstOrNullEntity, LongCount, Max, Min, Single, SingleOrDefault, SingleOrNullEntity,* and *Sum*. Examples of each are provided in the *API documentation* .