Contents

Inverting a query is accomplished through query inversion. Query inversion conveys a process by which we try to insure that any data queried from a backend datasource can also be queried, with the same results, by executing the same query against the local entity cache. This is difficult to insure under two conditions

1. Where a query involves a projection. (Select or SelectMany)
2. Where a query involves a subquery.

InversionMode

DevForce is able to automatically perform query inversion on queries that involve subqueries, but must rely on the developer's assistance in inverting projection queries via judicious use of *Include* clauses. Because of this it is often useful to construct queries using subqueries instead of projections in order to gain access to DevForce's automatic query inversion logic.

An example of query inversion follows:

The query "get me all Customers with Orders in the current year" will return references to Customer objects, but must first examine many Order objects in order to return the correct set of Customers. The query "give me the count of Customers located in Idaho" will return an integer, but must examine the Customer collection in the data source.

Query inversion is the process of retrieving those non-targeted objects that are nonetheless necessary for correct completion of a query. The most fundamental reason for doing query inversion is so that the query can be applied against a pool of data that combines unpersisted local data with data that exists in the datasource. This is, after all, what your end user normally wants: query results based on the state of the data as she has modified it.

The only place that combined pool of data can exist, prior to persisting changes, is the local cache. Therefore the query must ultimately be applied against the cache; and that operation, if it is to return correct results, requires the cache to contain all entities that must be examined in the course of satisfying the query. So to satisfy the query "get me all Customers with Orders in the current year", the cache must contain not only the Customers to which references will be returned, but also all of the current-year Orders that were placed by these customers.

Another way of understanding the idea of inverting queries is that the same query, if resubmitted during the same application session, can be satisfied entirely from the cache, without requiring another trip to the datasource.

A handy side effect of query inversion is that in practice there is a reasonably good chance that the related objects needed for satisfaction of the query will also be referenced in other ways by the application. In this very common scenario, the effect of the extra data retrieved is to improve client-side performance by eliminating the need for separate retrieval of the related objects.

Note that the end result of a query inversion process is very similar to that which occurs when the *.Include()* method is used in a query. Both processes result in the retrieval and local storage of objects that are related to a set of root objects that are the primary target of a particular query.

Four *InversionModes* are available in DevForce for a query:

	Implicit Instructions to DevForce
On	Attempt to retrieve, from the datasource and into the cache, entities other than the targeted type which are needed for correct processing of the query. If this attempt f exception.
Off	Do not attempt to retrieve entities other than the targeted type into the cache.
Try	Attempt to retrieve, from the datasource and into the cache, all entities other than the targeted type which are needed for correct processing of the query. However, if retrieve the entities of the directly targeted type, and do not throw an exception.
Manual	Do not attempt to invert the current query; but act as if it were successfully inverted (if it needed to be). You (the developer) should only use this *InversionMode* when you are prepared to guarantee, on your own, that the entity cache contains (or will contain, after the of the query operation) all the necessary related objects to return a correct result if submitted against the cache. Normally you would make good on this guarantee by retrieval operations (prior to the one in question) to retrieve the necessary related data; or by including calls to the *Include()* extension method in the current query, su necessary related data.

The default *InversionMode* is *Try*, and this will likely be your choice for most queries.

You should use *On* only if your application absolutely depends upon the related entities being brought into the cache by your query, and you should include exception handling in case the strategy fails.

Choose the *Off* setting if you only want the targeted entries retrieved into the cache. Be sure you choose a compatible FetchStrategy.

For queries that DevForce can successfully invert, the *InversionModes* of *Try* and *On* will yield the same end state: the query will be cached, and all related objects necessary to permit future satisfaction of the query entirely from the cache will be assumed to be present in the cache. If you use the *InversionMode* of *Manual* properly – that is, you take care to see that the necessary related objects get retrieved into the cache by some means or another before the query is submitted – then it, too, will produce the same ending state as the *Try* and *On* settings.

Queries that cannot be automatically inverted

The following types of queries cannot be automatically inverted:

- A query that returns a scalar result. This includes all aggregate queries (Count, Sum, Avg, etc.). Note that this group includes the example mentioned earlier in this discussion: “Give me the count of Customers located in Idaho.”

```
var query02 = _em1.Orders.Select(o => o.FreightCost).Sum();
```

```
Dim query02 = _em1.Orders.Select(Function(o) o.FreightCost).Sum()
```

- A query whose return type is a single element. These include queries that call *.First()*, *.Last()*, and *.Single()*

```
var query03 = _em1.Products.OrderByDescending(c =>
  c.ProductName).FirstOrNullEntity();
{{/code}}
```

```
Dim query03 = _em1.Products.OrderByDescending( _
  Function(c) c.ProductName).FirstOrNullEntity()
```

- A query whose return type is different from the type contained in the collection first referenced.

```
var query04 = _em1.Customers
  .Where(c => c.Country == "Argentina")
  .SelectMany(c => c.Orders);
```

```
Dim query04 = _em1.Customers _
  .Where(Function(c) c.Country = "Argentina") _
  .SelectMany(Function(c) c.Orders)
```