When you **prepare to go offline**, you will need to populate the client-side cache with whatever entities you will need before the database becomes unavailable. To do this, simply query for the entities that you need and they will automatically be brought into the cache. The *Include* operator in LINQ is very handy for this, as it allows you to fetch related entities in a single query.

For example, if you are caching all of today's Orders, you might also want to fetch the Customers, OrderDetails, and Products for those orders at the same time:

```
private void CacheTodaysOrders(EntityManager manager) {
  var query = manager.Orders.Where(o => o.OrderDate == DateTime.Today)
    .Include("Customer")
    .Include("OrderDetails.Products");
  query.ExecuteAsync(CacheTodaysOrdersComplete);
}
private void CacheTodaysOrdersComplete(EntityQueryOperation<Order> args) {
}
```

```
Private Sub CacheTodaysOrders(ByVal manager As EntityManager)
 Dim query = manager.Orders.Where(Function(o) o.OrderDate = Date.Today) _
    .Include("Customer") _
    .Include("OrderDetails.Products")
  query.ExecuteAsync(AddressOf CacheTodaysOrdersComplete)
End Sub
Private Sub CacheTodaysOrdersComplete(ByVal args As EntityQueryOperation(Of Order))
End Sub
```

If you are caching entities asynchronously, remember to wait for the operation to finish before going offline! (To learn more about fetching related entities in the same query, see Include related entities.)

There is no explicit limit to how much you can cache, but eventually memory, serialization, and query performance may be limiting factors. For large datasets (>50MB), you might consider using a local database such as SQL Server Express or SQL Server Compact Edition. (Note that you can create an EntityManager that connects to a local database by passing an *EntityServiceOption* of *UseLocalService* in the constructor.)