Lazy Loading

Another commonly misunderstood concept is that of lazy loading. Lazy loading can be a great help when we want to get to information at a later time. However, it can also introduce big overhead. If a query executes, and then a lazy-loaded object is requested another database query will execute to obtain that row information. In loops, this can be a large problem. Consider the following example.

var activeSchoolvar activeSchools = ctx.Schools.Where(s => s.IsActive).ToList();

var iowaSchools = activeSchools.Where(s => s.State.StateCode == "IA");s = ctx.Schools.Where(s => s.IsActive).ToList();

var iowaSchools = activeSchools.Where(s => s.State.StateCode == "IA");

This query doesn't look horrible. However, the devil is in the details. The first line will get all active schools, executing a SQL similar to SELECT \* FROM Schools WHERE IsActive = 1. This is what we expect; it only includes school information and limits. The problem comes in the next line. We have a navigation property on the School object linking to State; we then want to find items in a particular state.

That where clause will result in 1 query being executed for each row. If you have 1000 schools that are active, these two lines of code will result in 1,001 queries being executed. SearchSchools really bad, in the BadDataService of the provided sample, showcases this issue. When running the application, you will be able to see the query.

Fixing this issue is quite easy. If you make all where clauses before enumeration you will get around the issue. Alternatively, you can tell EF to eagerly load a Lazy Loaded item for a specific query. Changing to use below would result in a single query being used.

var activeSchools = ctx.Schools.Include(s => s.State)

var activeSchools = ctx.Schools.Include(s => s.State)

.Where(s => s.IsActive).ToList();

var iowaSchools = activeSchools.Where(s => s.State.StateCode == "IA"); .Where(s => s.IsActive).ToList();

var iowaSchools = activeSchools.Where(s => s.State.StateCode == "IA");

The Include(s => s.State) tells the query process to prepare for later usage.