LiteDB

Fork me on GitHub

DbRef

LiteDB is a document database, so there is no JOIN between collections. You can use embedded documents (sub-documents) or create a reference between collections. To create a reference you can use `[BsonRef]` attribute or use the `DbRef` method from the fluent API mapper.

Mapping a reference on database initialization

```
public class Customer
{
    public int CustomerId { get; set; }
    public string Name { get; set; }
}

public class Order
{
    public int OrderId { get; set; }
    public Customer Customer { get; set; }
}
```

If no custom mapping is created, when you save an `Order` , `Customer` is saved as an embedded document with no link to any other collection. Any changes made to documents in the `customers` collection will not be reflected in the `orders` collection.

```
Order => { _id: 123, Customer: { CustomerId: 99, Name: "John Doe" } }
```

If you want to store only a reference to a customer in `Order` , you can decorate your class:

```
public class Order
{
    public int OrderId { get; set; }

    [BsonRef("customers")] // where "customers" is the collection to be referenc
    public Customer Customer { get; set; }
}
```

Note that `BsonRef` decorates the full object being referenced, not an int `customerid` field that references an object in the other collection.

Or use fluent API:

```
BsonMapper.Global.Entity<Order>()
    .DbRef(x => x.Customer, "customers"); // where "customers" are Customer coll
```

Note: `Customer` needs to have a `[BsonId]` defined.

Now, when you store `Order` you are storing only the reference.

```
Order => { _id: 123, Customer: { $id: 4, $ref: "customers"} }
```

Querying results

When you query a document with a cross-collection reference, you can auto load references using the `Include` method before query.

```
var orders = db.GetCollection<Order>("orders");

var order1 = orders
    .Include(x => x.Customer)
    .FindById(1);
```

DbRef also support `List<T>` or `Array` , like:

```
public class Product
{
    public int ProductId { get; set; }
    public string Name { get; set; }
    public decimal Price { get; set; }
}

public class Order
{
    public int OrderId { get; set; }
    public DateTime OrderDate { get; set; }
    public List<Product> Products { get; set; }
}

BsonMapper.Global.Entity<Order>()
    .DbRef(x => x.Products, "products");
```

If the `Products` field is null or an empty list, the value will be preserved when being mapped from a `BsonDocument` to an `Order` . If you do not use `Include` in query, every `Product` in `Products` will be loaded with the id field set and all other fields null or default.

In v4, this include process occurs on BsonDocument engine level. It also support any level of include, just using `Path` syntax:

```
orders.Include(new string[] { "$.Customer", "$.Products[*]" });
```

If you are using `LiteCollection` or `Repository` you can also use Linq syntax:

```
// repository fluent syntax
db.Query<Order>()
    .Include(x => x.Customer)
    .Include(x => x.Products)
    .ToList();
```