

Transactions

As mentioned by default the `SubmitChanges` is wrapped in a `Transaction`, so all the changes are rolled back if one fails. The failures are often the result of the `Optimistic Concurrency` model that will throw a `ChangeConflictException`. if changes are detected since our application read its data. As an example consider the simple case where we are updating the `ShipCountry` of the `Orders` table:

OrderID	ShipCountry
10643	USA
10692	USA
10702	USA
10835	USA
10952	USA
11011	USA
11091	USA
11092	USA

We will use a `Breakpoint` in our code to simulate the changing of the data by another process after we have read it.

```
DataClasses1DataContext db = new DataClasses1DataContext();
var query = from o in db.Orders
            where o.Customer.CustomerID == "ALFKI"
            select o;
foreach (Order o in query)
{
    o.ShipCountry = "Canada";
}
db.SubmitChanges();
```

← All the Orders associated with "ALFKI" have USA as ShipCountry

← In "memory" change each ShipCountry to Canada

← Put a Breakpoint here



Concurrency and Transactions

From SQL Studio or DataBase Explorer we perform the following Update:

```
Update orders  
set shipcountry='poland'  
where (CustomerID = 'alfki' and orderid='10692')
```

So our table now looks like :

OrderID	ShipCountry
10643	USA
10692	Poland1
10702	USA
10835	USA
10952	USA
11011	USA
11091	USA
11092	USA

orderID 10692 is now changed
so when we execute the
Submitchanges we see the
following Exception:

```
db.SubmitChanges();
```

PE

```
using (TransactionSc  
{  
    try  
    {  
        db.SubmitCha  
    }  
    catch  
    {  
        int ii = 0;
```

ChangeConflictException was unhandled

1 of 2 updates failed.

Troubleshooting tips:

[Get general help for exceptions.](#)

[Search for more Help Online...](#)

Transactions

If we now look at the table it is unchanged including the OrderID 10643 which actually worked since the SubmitChanges was wrapped in a Transaction.

OrderID	ShipCountry
10643	USA
10692	USA
10702	USA
10835	USA
10952	USA
11011	USA
11091	USA
11092	USA

if we need to explicitly define the Transaction then TransactionScope can be used, this is especially useful if we are updating multiple tables or calling operations that involve other Transaction Resource managers like WCF.

```
try
{
    using (TransactionScope scope = new TransactionScope())
    {
        db.SubmitChanges();
        scope.Complete();
    }
    catch (exception e)
    {
        Console.Write(e.message);
    }
}
```

← Only Commit the changes if all the updates work

Make sure the Distributed Transaction Coordinator (DTC) is setup if your data base server is on another machine
<http://forums.microsoft.com/MSDN/ShowPost.aspx?PostID=230390&SiteID=1>

Transactions

Linq to SQL gives us a lot of control over the management of these Conflict Exceptions. We have the option to Continue the process on the remaining objects after a conflict is detected. We do this by using the ConflictMode Enum which is a SubmitChanges parameter.

```
using (TransactionScope scope = new TransactionScope())
{
    try
    {
        db.SubmitChanges(ConflictMode.ContinueOnConflict);
    }
    catch (Exception e)
    {
        Console.WriteLine(e.Message);
    }
    finally
    {
        scope.Complete();
    }
}
```

← 1 of 9 updates failed

← Commit the successful updates

OrderID	ShipCountry
10643	Canada
10692	Poland
10702	Canada
10835	Canada

All updates were performed except orderid 10692 that threw the exception

Transactions

We can also retry failing updates through the using the ChangeConflicts collection of the DataContext:

```
using (TransactionScope scope = new TransactionScope())
{
    try
    {
        db.SubmitChanges(ConflictMode.ContinueOnConflict);
    }
    catch
    {
        foreach (ObjectChangeConflict conflict in db.ChangeConflicts)
        {
            Order o = conflict.Object as Order;
            conflict.Resolve(RefreshMode.KeepCurrentValues);
        }
        db.SubmitChanges();
    }
}
```

*cast to Order
and retry
update*

*List of objects
with conflicts*

*overwrite database values
so Poland becomes
Canada*

OrderID	ShipCountry
10643	Canada
10692	Canada
10702	Canada
10835	Canada
10952	Canada