Linq To Entity を使ってみよう

えムナウ（児玉宏之）
http://mnow.jp/
http://mnow.wankuma.com/
http://blogs.wankuma.com/mnow/
http://www.ailight.jp/blog/mnow/

アジェンタ

- はじめに
- データベース設計をしてみよう
  - 納品書システムのデータベースを作る
- Linq to Entitiesで組んでみよう
- Linq to Entitiesマッピング シナリオ
- まとめ

はじめに

- リレーショナルデータベースを使ったシステムが普及して年月がたちます。
- オブジェクト指向言語での開発も普及して年月がたちます。
- しかし、その間には溝があります。
- オブジェクト指向プログラミング（OOP）とリレーショナルデータベース（RDB）の間の不一致で、O/R インピーダンスミスマッチと呼ばれるものです。

はじめに

- O/R インピーダンスミスマッチとはどんなもの
  - オブジェクト指向は継承ができるが、リレーショナルデータベースは継承ができない。
  - オブジェクト指向は関連情報をクラス内部にコレクションとして持てるが、リレーショナルデータベースは別のテーブルにデータをもってそのテーブルからリレーションを行う。
  - Linq to SQL も Linq to Entities もこの二つの O/R インピーダンスミスマッチを解決します。

はじめに

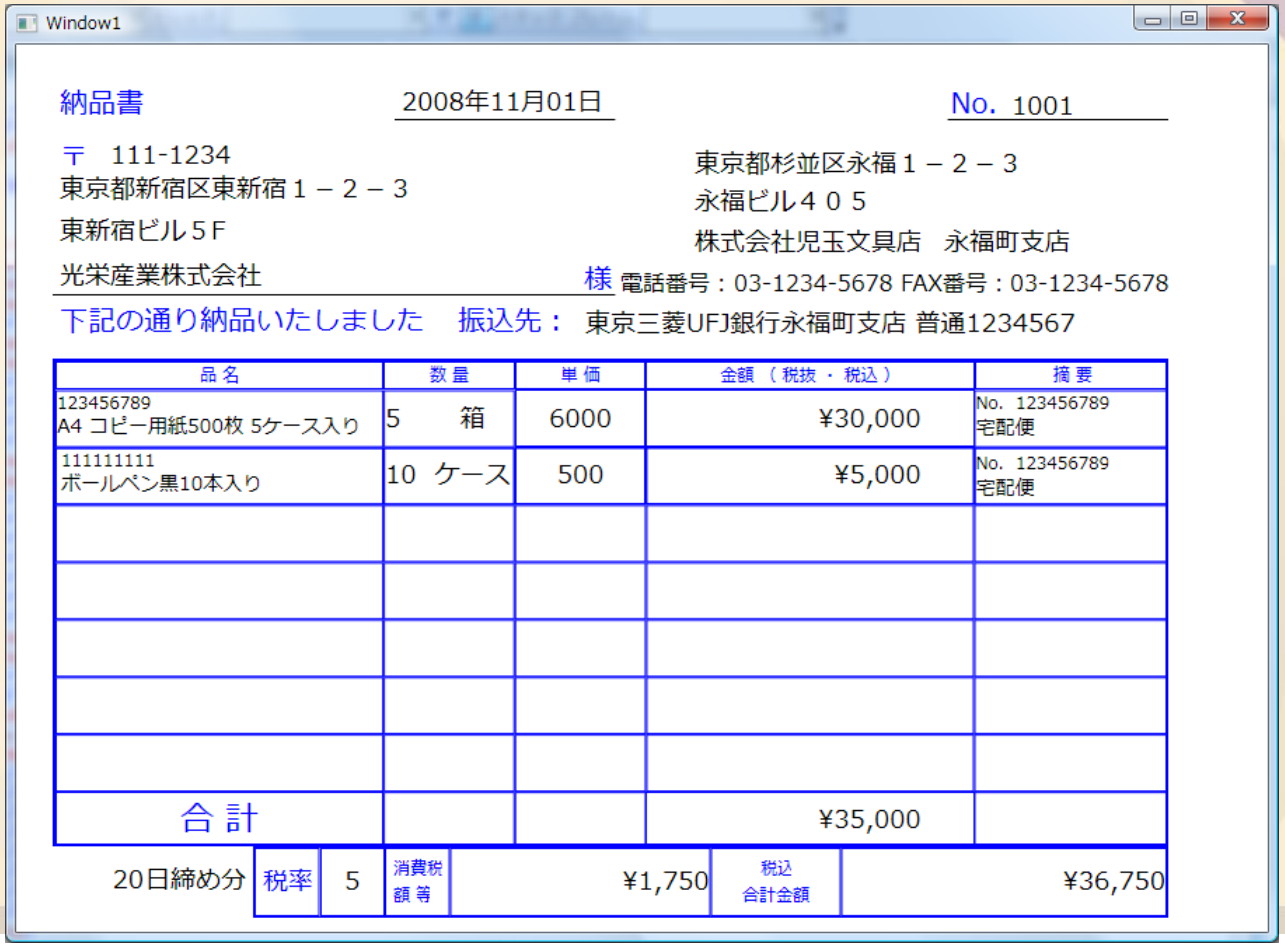
- 納品書を作成しながら、データベース設計とO/R インピーダンスミスマッチ を見てみましょう。
- Linq to Entities は、概念レベル記述のための多対多の関係・複数テーブルの結合というLinq to SQL にない記述も可能です。
- Linq to Entities で作成してみて、良さと欠点を見ていきましょう。

データベース設計をしてみよう

- データベース設計には3段階あります
  - 概念レベル設計
    - 対象となるシステム内に存在するエンティティおよびリレーションシップを定義
  - 論理モデル設計
    - 外部キー制約を用いながらエンティティおよびリレーションシップをテーブルとして正規化
  - 物理モデル設計
    - エンジン固有機能やパーティション分割やインデックス化など特定のデータ エンジンの機能に対応

データベース設計をしてみよう

- 納品書のデータベースを作る

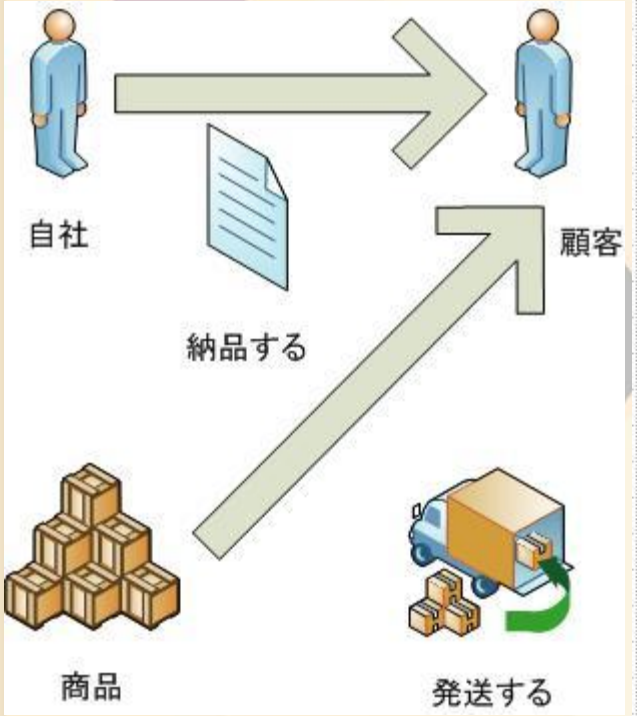


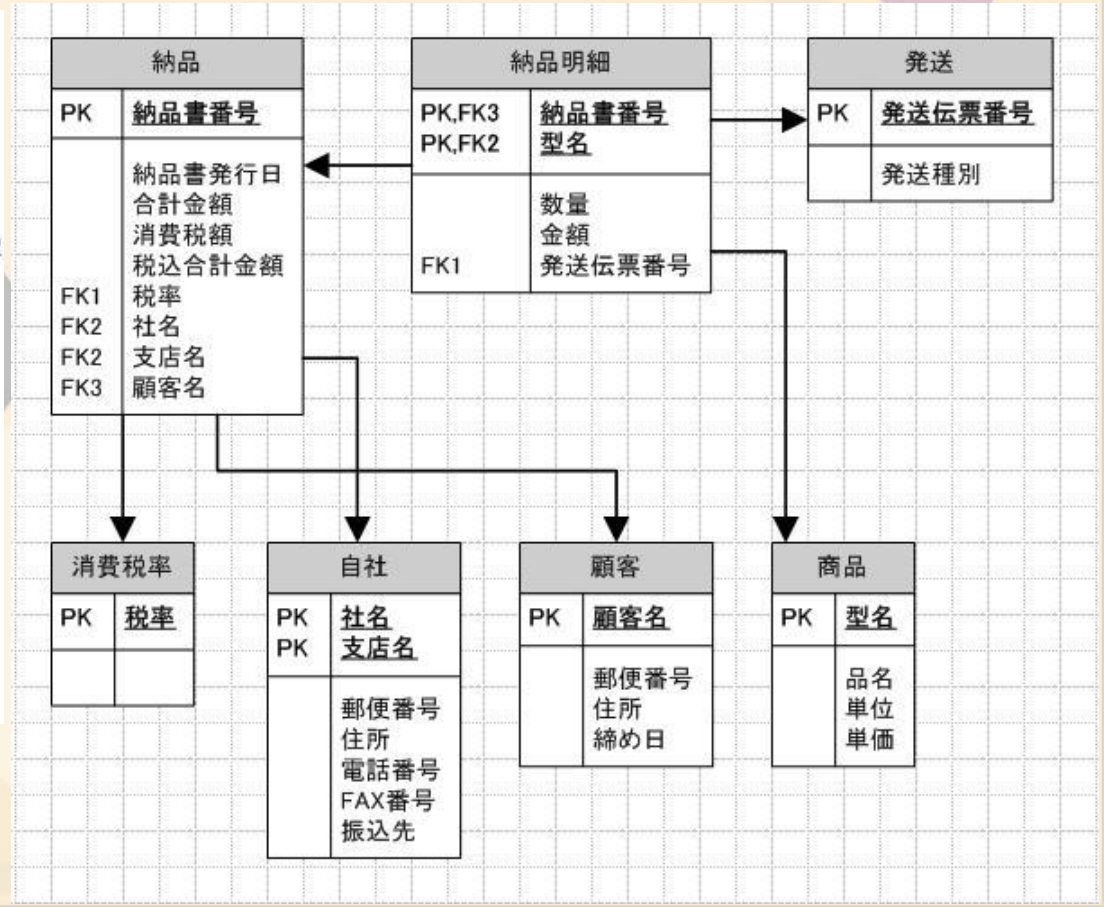
データベース設計をしてみよう

- トータルシステムを運用も含めモデル化
- トータルシステムの一部の納品書とは何か
  - いつ、どこから、どこへ、何を、いくつ、何で送り、いくらで、いつ払ってもらうかを書いたもの。
  - 納品年月日や納品書番号、自社情報、顧客情報、商品情報、数量、配送情報、合計金額、締め日が必要
  - 顧客情報は郵便番号、住所、会社名が必要
  - 商品情報、数量、配送情報は明細というくくりで
  - などなど、トップダウンで設計していく。

データベース設計をしてみよう

- 概念モデルができた





データベース設計をしてみよう

- Excelの表で書いてみよう

納品書発行日	納品書番号	顧客郵便番号	顧客住所	顧客名	自社郵便番号	自社住所	自社名
2008/11/1	1001	111-1234	東京都新宿区東新宿1－2－3 東新宿ビル5F	光栄産業	168-0064	東京都杉並区永福1－2－3 永福ビル405	児玉文具店
2008/11/1	1002	111-1234	東京都新宿区東新宿1－2－4	山田興産	168-0064	東京都杉並区永福1－2－3 永福ビル405	児玉文具店

自社支店名	自社電話番号	自社FAX番号	振込先	型名1	品名1	数量1	単位1	単価1
永福町	03-1234-5678	03-1234-5678	東京三菱UFJ銀行永福町支店 1234567	123456789	A4 コピー用紙500枚 5ケース入り	5	箱	6000
永福町	03-1234-5678	03-1234-5678	東京三菱UFJ銀行永福町支店 1234567	123456789	A4 コピー用紙500枚 5ケース入り	10	箱	6000

金額1	発送伝票番号1	発送種別1	型名2	品名2	数量2	単位2	単価2
30000	123456789	宅配便	111111111	ボールペン黒10本入り	10	ケース	500
60000	123456666	宅配便	222222222	ボールペン赤10本入り	20	ケース	500

金額2	発送伝票番号2	発送種別2	合計金額	締め日	消費税率	消費税額	税込合計金額
5000	123456789	宅配便	35000	20	5	1750	36750
10000	123457777	宅配便	70000	0	5	3500	73500

データベース設計をしてみよう

- 繰り返し部分を別にしてみよう
  - 明細部分 （ヘッダー部分はその残り）

属性	納品明細				
キー	納品書番号	1001	1001	1002	1002
キー	型名	123456789	111111111	123456789	222222222
	品名	A4 コピー用紙500枚 5ケース入り	ボールペン黒10本入り	A4 コピー用紙500枚 5ケース入り	ボールペン赤10本入り
	数量	5	10	10	20
	単位	箱	ケース	箱	ケース
	単価	6000	500	6000	500
	金額	30000	5000	60000	10000
	発送伝票番号	123456789	123456789	123456666	123457777
	発送種別	宅配便	宅配便	宅配便	宅配便

データベース設計をしてみよう

- キーに結びついた塊を外に出そう

属性	納品明細				
キー	納品書番号	1001	1001	1002	1002
キー、参照	型名	123456789	111111111	123456789	222222222
	数量	5	10	10	20
	金額	30000	5000	60000	10000
	発送伝票番号	123456789	123456789	123456666	123457777
	発送種別	宅配便	宅配便	宅配便	宅配便

属性	商品			
キー	型名	123456789	111111111	222222222
	品名	A4 コピー用紙500枚 5ケース入り	ボールペン黒10本入り	ボールペン赤10本入り
	単位	箱	ケース	ケース
	単価	6000	500	500

データベース設計をしてみよう

- 意味のある塊を外に出そう　ヘッダー部分

属性	納品		
キー	納品書番号	1001	1002
	納品書発行日	2008/11/1	2008/11/1
参照	顧客名	光栄産業	山田興産
参照	自社名	児玉文具店	児玉文具店
参照	自社支店名	永福町	永福町

属性	顧客		
キー	社名	光栄産業	山田興産
	郵便番号	111-1234	111-1234
	住所	東京都新宿区東新宿１－２－３ 東新宿ビル5F	東京都新宿区東新宿１－２－４
	締め日	20	0

属性	自社	
キー	社名	児玉文具店
キー	支店名	永福町
	郵便番号	168-0064
	住所	東京都杉並区永福１－２－３ 永福ビル405
	電話番号	03-1234-5678
	FAX番号	03-1234-5678
	振込先	東京三菱UFJ銀行永福町支店 1234567

属性	消費税率	
キー	消費税率	5

計算で求める部分も同時に削除

データベース設計をしてみよう

- 意味のある塊を外に出そう　明細部分

属性	納品明細				
キー	納品書番号	1001	1001	1002	1002
キー、参照	型名	123456789	111111111	123456789	222222222
	数量	5	10	10	20
参照	伝票番号	123456789	123456789	123456666	123457777

計算で求める部分も同時に削除

属性	商品			
キー	型名	123456789	111111111	222222222
	品名	A4 コピー用紙500枚 5ケース入り	ボールペン黒10本入り	ボールペン赤10本入り
	単位	箱	ケース	ケース
	単価	6000	500	500

属性	発送			
キー	伝票番号	123456789	123456666	123457777
	発送種別	宅配便	宅配便	宅配便

データベース設計をしてみよう

- 元データを表にする
  - テーブル化
- 繰り返し部分を別にしてみよう
  - 第1正規形
- キーに結びついた塊を外に出そう
  - 第2正規形
- 意味のある塊を外に出そう
  - 第3正規形
- 論理モデルができた

データベース設計をしてみよう

- マスターとトランザクションに分類する。
  - マスターは物や事柄の名前で名詞が基本。
    - 自社、商品、顧客、消費税率
    - 削除は行わない、有効期限や削除済みフラグを持つ。
    - 作成者と最終更新者や更新履歴をもつか考慮する。
    - 排他制御を考慮する。
  - トランザクションは「する」がついて動詞化できる。
    - 納品、納品明細、発送
    - 発生時刻を明確化、必要なら完了時刻も付加する。
    - 修正や削除を行わない、赤伝票を発行する。

データベース設計をしてみよう

- SQLサーバー用にカスタマイズする
  - 分割したほうが扱いやすいものは分割する。
    - 住所は伝票では2段なので2つに分ける。
  - 客が、変わらないからキーを納品書番号にしろと言われても信じない、IDのキーを別に設ける。
  - 排他処理は先更新を有効、後更新をエラーにするなら、timestampが便利。
- データベースの配置やインデックスなど決めていく。
- 物理モデルができた。

データベース設計をしてみよう

- マスタ

属性	自社	データ型	Null許容
PK	ID	int	
	社名	nvarchar(MAX)	
	支店名	nvarchar(MAX)	TRUE
	郵便番号	nvarchar(MAX)	TRUE
	住所1行目	nvarchar(MAX)	TRUE
	住所2行目	nvarchar(MAX)	TRUE
	電話番号	nvarchar(MAX)	TRUE
	FAX番号	nvarchar(MAX)	TRUE
	振込先	nvarchar(MAX)	TRUE
	開始日時	datetime	TRUE
	終了日時	datetime	TRUE
	timestamp	timestamp	

属性	顧客	データ型	Null許容
PK	ID	int	
	社名	nvarchar(MAX)	
	郵便番号	nvarchar(MAX)	TRUE
	住所1行目	nvarchar(MAX)	TRUE
	住所2行目	nvarchar(MAX)	TRUE
	締め日	int	TRUE
	開始日時	datetime	TRUE
	終了日時	datetime	TRUE
	timestamp	timestamp	

属性	消費税率	データ型	Null許容
PK	ID	int	
	消費税率	float	
	開始日時	datetime	TRUE
	終了日時	datetime	TRUE
	timestamp	timestamp	

属性	商品	データ型	Null許容
PK	ID	int	
	型名	nvarchar(MAX)	
	品名	nvarchar(MAX)	
	単位	nvarchar(MAX)	TRUE
	単価	float	
	開始日時	datetime	TRUE
	終了日時	datetime	TRUE
	timestamp	timestamp	

データベース設計をしてみよう

- トランザクション

属性	納品	データ型	Null許容
PK	ID	int	
	納品書番号	int	
	納品書発行日	datetime	
FK	顧客ID	int	
FK	自社ID	int	
FK	消費税率ID	int	
	登録日時	datetime	

属性	納品明細	データ型	Null許容
PK	ID	int	
FK	納品ID	int	
FK	商品ID	int	
	数量	float	
FK	発送ID	int	TRUE
	登録日時	datetime	

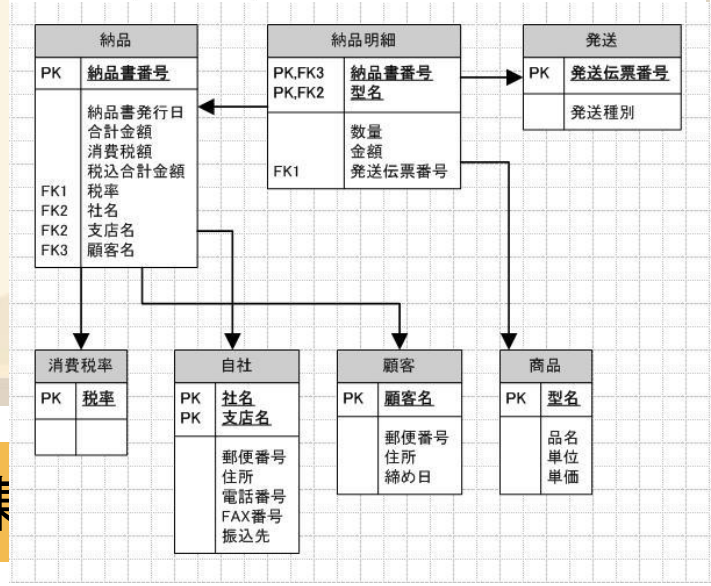
属性	発送	データ型	Null許容
PK	ID	int	
	伝票番号	nvarchar(MAX)	
	発送種別	nvarchar(MAX)	TRUE
	登録日時	datetime	

データベース設計をしてみよう

- データベースダイアグラム

Linq to Entities で組んでみよう

- Linq to Entitiesのモデルを作る



Linq to Entities で組んでみよう

- バイディングするためのコレクションの用意

```
namespace 納品書
{
    partial class 納品
    {
        private ObservableCollection<納品明細> _納品明細List;
        public ObservableCollection<納品明細> 納品明細List
        {
            get
            {
                return _納品明細List;
            }
        }

        private void 納品明細List作成()
        {
            if (_納品明細List == null && this.納品明細 != null)
            {
                _納品明細List = new ObservableCollection<納品明細>();
                foreach (納品明細 item in this.納品明細)
                {
                    item.金額作成();
                    item.PropertyChanged += new PropertyChangedEventHandler(納品明細_PropertyChanged);
                    _納品明細List.Add(item);
                }
            }
        }
    }
}
```

ObservableCollection を作る

合計計算のために
PropertyChanged
イベントを拾う

納品明細の参照を追加する

Linq to Entities で組んでみよう

- 計算列を partial でコーディング

```
namespace 納品書
{
    partial class 納品
    {
        private double _合計金額;
        public double 合計金額
        {
            get { return _合計金額; }
            set { OnPropertyChanging("合計金額"); _合計金額 = value;
OnPropertyChanged("合計金額"); }
        }
        private double _消費税額;
        public double 消費税額
        {
            get { return _消費税額; }
            set { OnPropertyChanging("消費税額"); _消費税額 = value;
OnPropertyChanged("消費税額"); }
        }
        private double _税込合計額;
        public double 税込合計額
        {
            get { return _税込合計額; }
            set { OnPropertyChanging("税込合計額"); _税込合計額 =
value; OnPropertyChanged("税込合計額"); }
        }
}
```

```
    partial class 納品明細
    {
        private double _金額;
        public double 金額
        {
            get { return _金額; }
            set { OnPropertyChanging("金額"); _金額 = value;
OnPropertyChanged("金額"); }
        }
    }
}
```

OnPropertyChanging や OnPropertyChanged を呼ぶ

Linq to Entities で組んでみよう

- 計算列の実際の計算

PropertyChanged イベント

```
namespace 納品書
{
    void 納品明細_PropertyChanged(object sender,
PropertyChangedEventArgs e)
    {
        if (e.PropertyName == "金額")
        {
            合計計算();
        }
    }
    private void 合計計算()
    {
        if (this.納品明細 == null || this.消費税率 == null) return;

        double sum = 0.0;
        foreach (納品明細 item in this.納品明細)
        {
            sum += item.金額;
        }
        this.合計金額 = sum;
        this.消費税額 = sum * this.消費税率.消費税率1 / 100.0;
        this.税込合計額 = this.合計金額 + this.消費税額;
    }
```

```
    public void 明細作成()
    {
        納品明細List作成();
        合計計算();
    }
}
partial class 納品明細
{
    partial void On数量Changed()
    {
        金額作成();
    }
    public void 金額作成()
    {
        if (this.商品 != null)
        {
            this.金額 = this.商品.単価 * this.数量;
        }
    }
}
```

OnXXXChanged

Linq to Entities で組んでみよう

- メインで画面ロード時にデータを読み込む

```
namespace 納品書
{
    public partial class Window1 : Window
    {
        納品システムEntities 納品システム = new 納品システムEntities();
        public Window1()
        {
            InitializeComponent();
        }
        private void Window_Loaded(object sender, RoutedEventArgs e)
        {
            var query = from 納品テーブル in 納品システム.納品
                        .Include("納品明細")
                        .Include("顧客")
                        .Include("自社")
                        .Include("消費税率")
                        .Include("納品明細.商品")
                        .Include("納品明細.発送")
                        where 納品テーブル.ID == 1
                        select 納品テーブル;
            ObjectDataProvider 納品DS = (ObjectDataProvider)this.FindResource("納品DS");

            納品DS.ObjectType = null;
            納品 data = query.First<納品>();
            data.明細作成();
            納品DS.ObjectInstance = data;
        }
    }
}
```

.Include で関係する Entity を一緒に読むようにする

Blend でUIを作るには、Resource に ObjectDataProvider で入れたほうが便利

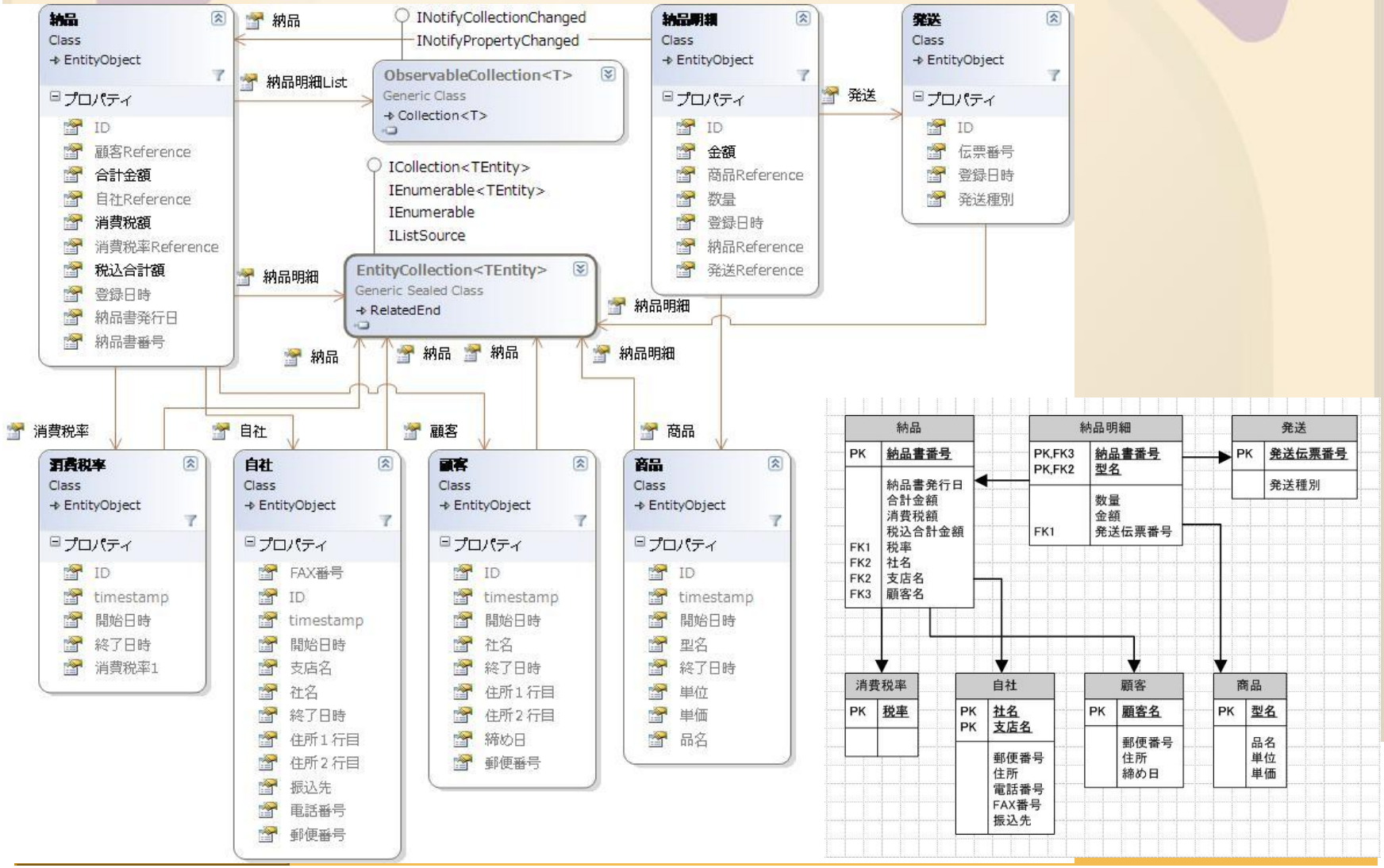
納品明細List を作成して計算列を計算

Linq to Entities で組んでみよう

- UIを作成してLabelにデータをバインド
  - <ObjectDataProvider x:Key="納品DS" ObjectType="{x:Type 納品書:納品}" d:IsDataSource="True"/>
  - Content="{Binding Path=納品書番号, Source={StaticResource 納品DS}}"
  - Content="{Binding Path=顧客.住所1行目, Source={StaticResource 納品DS}}"
  - Content="{Binding Path=自社.社名, Source={StaticResource 納品DS}}"
  - Content="{Binding Path=納品明細List[0].商品.型名, Source={StaticResource 納品DS}}"
  - Content="{Binding Path=納品明細List[0].発送.発送種別, Source={StaticResource 納品DS}}"
  - Content="{Binding Path=税込合計額, Source={StaticResource 納品DS}}"

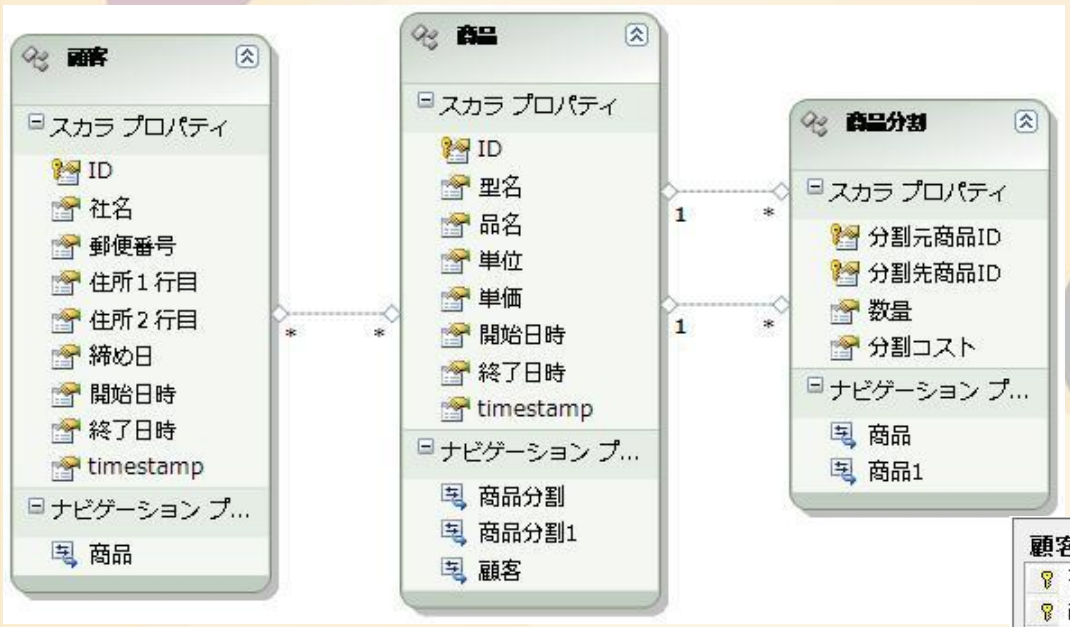
Linq to Entities で組んでみよう

- クラスダイアグラム



Linq to Entities マッピング シナリオ

- 多対多の関係

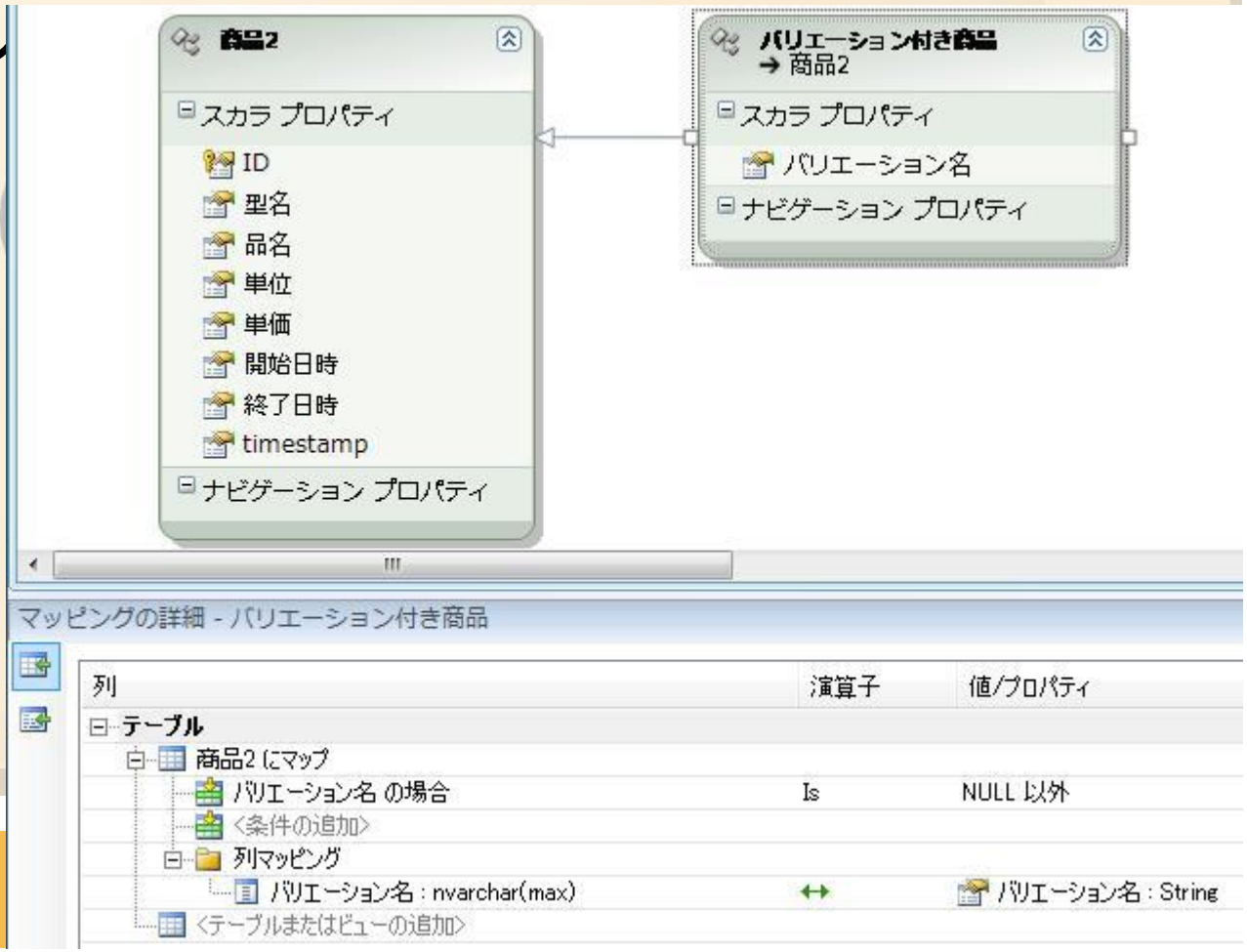


Linq to Entities マッピング シナリオ

- Table-Per-Type 継承
  - 固有の情報を格納し、共通のキーを共有している場合に商品を抽象型にして継承する。

Linq to Entities マッピング シナリオ

- Table-Per-Hierarchy継承
  - 階層型クラス継承、商品2を抽象型にしてバリエーションあるものを抽出



わんくま同盟

Linq to Entities マッピング シナリオ

- Multiple-Entity-Sets-per-Type
  - 一つの「支店別売上」型を複数の地方でEntityとして利用する。
  - 型の定義は一つで、インスタンスは複数。
  - 現在は残念ながら、モデルビューアーが、サポートしていないが、XMLで書けば出来る。

まとめ

- Linq to Entities は、O/R インピーダンスミスマッチと概念レベルと論理レベルのミスマッチを解消してくれます。
- 多対多の関係や様々な継承関係でオブジェクト指向で設計された概念レベルにより近い Entity を利用できます。
- 初期バージョンなので不便なところは Feed Back して修正してもらいながら、どんどん使っていきましょう。