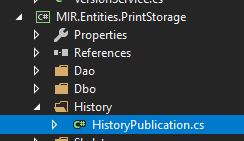
# BLToolkit

Business Logic Toolkit is a set of components to simplify .NET application development. BLToolkit is provided as source code that you can use "as is" or customize for your applications.

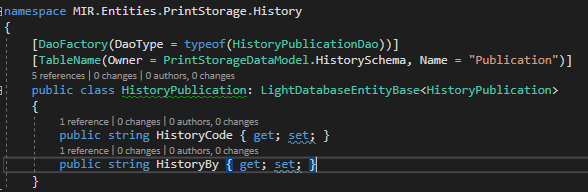
<https://github.com/igor-tkachev/bltoolkit/wiki>

# Vytvoreni daoTridy

Mam tabuli v databazi, ve schemadu History, jmenuje se History.Publication. Musim ji namapovat. Ve Skenovadle mam projekt MIR.Entities.PrintStorage. Jsou tu slozky Dao – daoTridy, Dbo – entity ze schematu dbo, History – entity ze schematu History.



Vytvoril jsem teda tridu HistoryPublication. Pridam ji atributy, podle kterych BLToolkit vi, ve kterem schematu a jakou tabuli hledat a jake dao se pro tuto entitu pouziva. Trida nema primarni klic, proto se dedi od LightDataBaseEntityBase<HistoryPublication>.



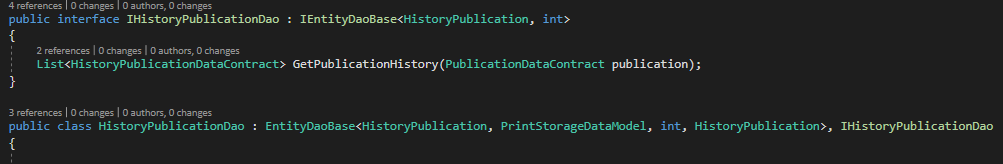
Schema a tabule se musi pridat na PrintStorageDataModel:

public const string HistorySchema = "History";

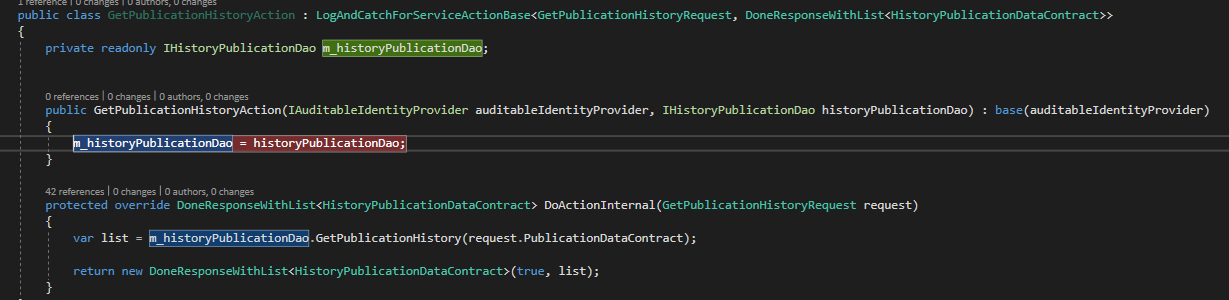
public Table<HistoryPublication> HistoryPublication => GetTable<HistoryPublication>();

Vytvorim HistoryPublicationDao. Presto, ze tabule v db nema primarni klic, pouziju EntityDaoBase a k nemu rozhrani podedene od:

IEntityDaoBase<T, TKey> (doporučený předek pro všechny dao interface) o stejné metody jako má transaction manager. (Mitroz).



DaoTridu si resolvnu nekde v konstruktoru a na ni muzu volat metody z Interfacu



Melo by fungovat.

# Prevedene poznamky z .rtf taky vesmir

**‎8/‎9/‎2017 BLTOOLKIT**

* Nainstalujeme si z firemniho serveru BLToolkit.CP.4.dll (Package source Nielsen), zaskrtneme projekty ve kterych budeme bl pouzivat (WPFUniverse).
* Do App.config v projektu kde je MainWindow.xaml , nastavime connectionString. (vezmeme ho z vlastnosti databaze ).
* Musime priradit tridy entit k tabulkam pro Galaxie.cs : [TableName("Galaxie", Owner = "dbo")] (Patri to nad Class)
* Musime nastavit pro sloupec Id , ze se jedna o primarni klic a automatickou inkrementaci [PrimaryKey] [Identity]
* Dao trida se zmeni uz nebude dedit od EntityDaoBase ani pouzivat jeji metody na pristup do db.
* Vytvorime novou chytrou tridu , bude se jmenovat UniverseDataModel. Placneme je do WpfUniverseEntities.
* UniverseDataModel dedi od DbManager , potrebuje using BLToolkit.Data a BLToolkit.Data.Linq.
* Rekneme ji ze existuje nejaka tabulka ktera se jmenuje Galaxie tak, ze si do public atributu vratime metodou GetTable() tabulku galaxii.
* Ve tride Galaxy dao si napiseme metody pro nacteni vsech galaxii a ulozeni galaxie.
* Do usingu si vlozime UniverseDataModel , kde bude SQLQuery. Pokud query sami nedefinujeme , vrati updatne nebo insertne vsechny sloupce.
* Query muzeme napsat i sami : model.VlastnostiPlanet.Where(x => x.VlastnostId == vlastnost.Id).Delete();
* Pri CRUD operacich kde jich probiha vic najednou, priklad tlacitko na checknuti vsech checkboxu naraz, je potreba pouzit transakci.

Pri transakci musime metode na dao tride ktera provadi zapis, dat zavislost na instanci UniverseDataModel a crud operaci provest prave na teto instanci.

Ve tride VlastnostiPlanetDao je metoda

public void RemovePropertyFromPlanet(int selectedPlanetId, int vlastnostId, UniverseDataModel model){

model.VlastnostiPlanet.Where(x => x.PlanetaId == selectedPlanetId && x.VlastnostId == vlastnostId).Delete();

}

Ktera je volana ve tride VlastnostDao a prave model ktery vznikne tady je parametrem na kterem se operace provadi. Jinak by se vytvorila nova instance modelu a transakce by probehla napul .

* Metody DoCheckAll a DoUncheckAll jsme zrefaktorovali do metody DoAllChanges protoze meli stejny kod.
* Odregistrovali jsme se z eventu OnPropertySelectChanged , aby nemohlo probehnout kliknuti na checkbox (a tim vyvolani eventu ktery uklada do databaze ) od uzivatele.
* V usingu kde vznikne objekt UniverseDataModel model, do konstrukce try{} catch{} vlozime transakci : model.BeginTransaction(); tady je jasne , ze transakce probiha prave na tomto modelu , proto ho predavame dao tride jako ten model na kterem se uklada.
* Osetrime jestli je nebo neni checknuto aby jsme nevkladali neco co uz existuje => PrimaryKey
* SqlException: Violation of PRIMARY KEY constraint 'PK\_VlastnostiPlanet'. Cannot insert duplicate key in object 'dbo.VlastnostiPlanet'. The duplicate key value is (713, 1). VAZEBNI TABULKA SE VKLADA UZ PRI KLIKNUTI NA CHECKBOX A JA JSEM JI CHTEL VLOZIT ZNOVU
* MUSI SE PROVEST KONTROLA , JESTLI UZ NENI VLASTNOST CHECKNUTA
* A Muzeme insertovat nebo deletovat.
* Abychom nezapomneli na opetovne zaregistrovani eventu (kliknuti na checkbox) celou zalezitost vlozime do try - finally (bez catch ) , kde ve finally provedeme opetovnou registraci.
* Provedeme v nejakem cyklu napr foreachi operace na databazi , po skonceni musime zmeny Commitnout : model.CommitTransaction();
* Operace su musi provest i na SelectedPlanet.Properties kvuli bindingu.
* A v catchi zachytime vyjimku kdyz vznikne vyjimka kterou neni mozno osetrit , napr spadne spojeni s databazi . Provede se model.RollbackTransaction aby se zmeny neprovedly napul.
* V takovem pripade pustime vyjimku dal pomoci throw a nechame aplikaci spadnout.

Na ladeni se pouziva CallStack , pri vyhozeni vyjimky Debug => windows => CallStack.

**‎10 ‎August, ‎2017**

* Jakym zpusobem se nastavuje Id pro novou vlastnost ? Zeptame se databaze na max moznou hodnotu return model.Vlastnost.Max(x => x.Id);
* Refresh funguje pokud mame jen jednu instanci kterou upravujeme ve vice oknech . Tady je trida Manager ktery je static v MainWindow a kolekce ListVlastnosti na tride Manager se predava a upravuje po cele aplikaci.

**‎11 ‎August, ‎2017**

* Aby aplikace nebyla zaspagetovana zavislostma v konstruktorech na ruzne viewModely , vytvorime si interface IGalaxySelector atd., kde soucast rozhrani bude event a vyhazovaci metoda. Tridam se zmeni zavislosti na viewModelech a misto nich dostanou zavislost na interfacu, pricemz se jim preda jen to co vlastne potrebujou a to jsou eventy ktere informuji , ze se zmenila galaxie , planeta atd.
* Timto zpusobem je mozne v aplikaci vymenit viewModely za jine , staci aby taky implementovali rozhrani INecoSelector.
* Doma se podivat na databazove transakce , zatim nas zajima readCommited (to je o politice zamku v databazich )
* Odspagetujeme aplikaci tim ze pro vsechny zavislosti vytvorime rozhrani a predavame uz jen to.
* Odregistrovali jsme v DoCheckAll event kliknuti , zapsali jsme vsechno nebo nic a zaregistrovali zpet. V tomto miste se musi pridat vlastnosti do SelectedPlanet.Properties , jinak do databaze se vsechno ulozi , ale po prekliknuti planety tam a zpet nebude view vedet o tom , ze se neco zmenilo.
* **BLToolkit pokracovani:**
* Pomoci nugetManageru - Solution -> ManageNuGet Packages for Solution -> nainstalujeme MediaResearch.Framework.DataAccess.BLToolkit . Uz mame z drivejsku nainstalovane BLToolkit.CP.4.dll., zaskrtnute vsechny assembly (Core, Entity ...)
* V BLToolkitu jsou tridy ktere za nas umi vytvaret dao objekty . Tridy entit si napiseme sami, v zavorkach nad entitou [DaoFactory(DaoType = typeof(GalaxyDao))] jsout tzv. metadata.
* Do tridy entit pridame usingy : using Mediaresearch.Framework.DataAccess.BLToolkit; using Mediaresearch.Framework.DataAccess.BLToolkit.DaoFactory;
* Entita ktera ma identitu v databazi bude dedit od public class Galaxie : LightDatabaseEntityIdentityIntKey<Galaxie>
* V metadatech ma informace [TableName("Galaxie", Owner = "dbo")] , kde informuje o jmenu databaze pro pripade ze by aplikace pracovala soucasne s vice databazema.

[DaoFactory(DaoType = typeof(GalaxyDao))] ., je na to , aby se vedelo jaky typ dao objektu se ma z entity vytvorit.

* Trida UniverseDataModel musi dedit od TransactionDbManager, ma 8 konstruktoru vetsina z nich vola konstruktor rodicovske tridy.
* Z app configu si BLToolkit umi vzit ConnectionString
* V MainWindow rekneme tride EntityDaoFactory ve ktere Assembly budeme brat daoObjekty : Assembly entities = Assembly.GetAssembly(typeof(Galaxie));
* Vytvorime chytrou tridu EntityDaoFactory , rekneme ji v konstruktoru , ze databaze se jmenuje "universe" a kde ma vzit ConnectionString.

EntityDaoFactory daoFactory = new EntityDaoFactory("universe", Settings.Default.ConnectionDb)

Ten si vezme z App.config

Tridy jako je tahle muzeme v MainWindow tvorit protoze mame nainstalovany BLToolkit a using na

using Mediaresearch.Framework.DataAccess.BLToolkit.DaoFactory; a

using Mediaresearch.Framework.DataAccess.BLToolkit.Dao;

* Vytvorime rozhrani IDaoSource daoSource = new EntityDaoFactoryDaoSource(daoFactory); Je to zdroj daoObjektu. Aby jsme se co nejvic odspagetovali budeme pouzivat vsude rozhrani , takze misto pouzivani GalaxyDao objektu budeme pouzivat jeho rozhrani IGalaxyDao.
* Trida GalaxyDao bude nove dedit od : EntityIdentityKeyDaoBase<Galaxie, UniverseDataModel, int , Galaxie> a implementovat IGalaxyDao. Ma dva konstruktory, zatim vim , ze se v nich predava string dBAlias , aby jsme vedeli ve ktere databazi jsme.

Uvnitr si rovnou napiseme rozhrani IGalaxyDao ktere dedi od : ISimpleDao<Galaxie, int >

============================================================

* **Ukoly:**
* Odendat SelectedPlanet z MainWindow
* Predelat tridy entit podle galaxie a dao objekty podle galaxyDao
* Predelat zavislosti ve viewModelech na IDaoSource
* Dao vezmeme pomoci metody m\_galaxyDao = daoSource.GetDaoByEntityType<IGalaxyDao, Galaxie, int > ();

===========================================================

* PropertiesViewModel chce vedet ktera planeta byla vybrana aby podle ni mohl checknout boxy a po skrtnuti mohl zapsat nebo odebrat vlastnost z vazebnich tabulek. SelectedPlanet objekt dostane od eventu OnPlanetChanged. Dostane event proto, ze ma zavislost na rozhrani IPlanetSelector ktere mu umozni stat se posluchacem

m\_IPlanetSelector.OnPlanetChanged += OnPlanetChanged;

* Tady nam staci udelat pro kazdou checknutou polozku : vdc.IsChecked = true . To je vsechno o zapis se postara trida VlastnostDataContract , tim , ze vyhodi event OnPropertySelectChanged. Kdo ho posloucha ? Metoda ve tride PropertiesViewModel . Ta v eventu dostane datacontract ktery se zmenil a podle toho jestli je checknutej ho zapise do databaze a do objektu na ktery je bindovano: SelectedPlanet.Properties
* To same se musi stat ve tride EditProperty kde ja blbec v konstruktoru tvorim novou selectedPlanet a proto mi to blbne.
* Tridu EditProperty jsem jen zaregistroval jako posluchace eventu OnPlanetChanged , jedine co dela je ze rekne : SelectedPlanet = selectedPlanet (predany objekt eventem). Aplikace pada na DoCheckAll => vyhazuje NullRefference , kdyz se pojede od breaku po vyjimce dostaneme se k oknu TransactionDbManager not found.
* Tak mne napada ze event OnPlanetChanged byl vyhozen jeste predtim nez jsem vytvoril objekt EditPropertyViewModel . Ten nemuze vedet jaka planeta byla vybrana protoze se zaregistruje jako posluchac az po vyhozeni eventu . Jak teda vi jakou planetu zobrazuje a kde ji vzal ? Musi dostat stejny objekt ktery v eventu dostala trida PropertiesViewModel. Ta uz totiz v case vyhozeni eventu existovala a objekt SelectedPlanet od eventu dostala. Mozna by to slo nejak z teto tridy dostat do EditViewModelu.

**‎14 ‎August, ‎2017**

* Entita Galaxie dedi od: LightDatabaxeEntityIdentityIntKey<Galaxie> . Protoze ma Identitu v databazi . Ma taky nastavenou inkrementaci.
* Entita Planeta to same: LightDatabaxeEntityIdentityIntKey<Planeta>
* Entita Vlastnost dedi od: LightDatabaseEntityIntKey<Vlastnost> Ma sice primarni klic ale nema nastavenou automatickou inkrementaci.
* Entita VlastnostiPlanet dedi od: LightDatabaseEntityMultiKeyBase<VlastnostiPlanet> Protoze ma dva cizi klice a nema automatickou inkrementaci. Je to jen vazebni tabulka.
* Dao trida : GalaxyDao dedi od : EntityIdentityKeyDaoBase <Galaxie , UniverseDataModel, int, Galaxie> Protoze ma identitu. Imlementuje IGalaxyDao . V tomto rozhrani jsou definovane metody ktere musi dao trida implementovat . Jsou tu metody na crud operace . Ty atomicke se neimplementujou , pouzivaji se rodicovske metody.
* Dao trida : PlanetDao dedi od : EntityIdentityKeyDaoBase <Planeta, UniverseDataModel, int, Planeta> Protoze ma identitu. Implementuje IPlanetDao. V tomto rozhrani jsou definovane metody ktere musi dao trida implementovat . Jsou tu metody na crud operace . Ty atomicke se neimplementujou , pouzivaji se rodicovske metody.
* Dao trida : VlastnostDao dedi od : EntityDaoBase <Vlastnost, UniverseDataModel, int, Vlastnost> Protoze **nema** identitu. Implementuje IVlastnostDao. V tomto rozhrani jsou definovane metody ktere musi dao trida implementovat . Jsou tu metody na crud operace . Ty atomicke se neimplementujou , pouzivaji se rodicovske metody.
* Dao trida : VlastnostiPlanetDao dedi od EntityMultiKeyDaoBase<VlastnostiPlanet, UniverseDataModel , VlastnostiPlanet> Protoze **nema** primarni klic. . Implementuje IVlastnostiPlanet. V tomto rozhrani jsou definovane metody ktere musi dao trida implementovat . Jsou tu metody na crud operace . Ty atomicke se neimplementujou , pouzivaji se rodicovske metody.
* Rozhrani IGalaxyDao dedi od ISimpleDao<Galaxie, int> (Entita, primarni klic)
* Rozhrani IPlanetDao dedi od ISimpleDao<Planeta, int>
* Rozhrani IVlastnostDao dedi od ISimpleDao<Vlastnost, int>
* Rozhrani IVlastnostiPlanet dedi od IEntityMultiKeyDaoBase < VlastnostiPlanet> jde totiz o vazebni tabulku ktera nema identitu a ma vic klicu.
* Tridu VlastnostDao dedi od EntityIdentityKeyDaoBase<Vlastnost, UniverseDataModel, int, Vlastnost> i kdyz nema inkrementaci , ale ma primarni klic
* Ma dva konstruktory (stejne jako ostatni dao tridy), ktere volaji konstruktor rodice, predavaji nazev databaze jako dbAlias a druhy konstr. urcuje navic stupen izolace transakci.
* Trida VlastnostiPlanetDao dedi od EntityMultiKeyDaoBase<VlastnostiPlanet, UniverseDataModel, VlastnostiPlanet> protoze ma jen dva cizi klice a nema identitu . Rozhrani IVlastnostiPlanetDao ale potom nemuze dedit od ISimpleDao<VlastnostiPlanet, int> protoze rodicovske rozhrani by chtelo primarni klic.

Proto rozhrani IVlastnostiPlanetDao dedi od IEntityMultiKeyDaoBase <VlastnostiPlanet> . Kvuli transakcim pridame do tohoto rozhrani metodu UniverseDataModel CreateDbModel(); (Trida VlastnostiPlanetDao ji musi implementovat, protoze implementuje rozhrani).

* Rozhrani se pise az za zavorky, ktere uzaviraji dao tridu . (Jinak to chce using static celou tridu)
* Ve vsech metodach kde bylo UniverseDataModel model = new UniverseDataModel, se prava cast zmeni na : using (UniverseDataModel model = CreateDbContext())

Tato metoda se stara mimo jine o transakce

* Sloupec Id v entitach byt nemusi protoze je v predkovi.
* Ve tride ktera se stara o vazebni tabulku bude v rozhrani navic metoda ktera nam vrati UniverseDataModel v podobe DbContext.

public UniverseDataModel CreateDbModel() { return CreateDbContext(); }

Tato metoda nam vrati instanci kterou pouzijeme pro zapis transakce.

* V metode DoAllChanges uz take netvorime UniverseDataModel ale pouzijeme metodu na tride VlastnostiPlanet CreateDbModel.

**Automaticke transakce pomoci TransactionManageru**

* V MainWindow si do vytvareni instance EntityDaoFactory pridame dalsi parametr - je to nova instance TransactionManageru.

EntityDaoFactory daoFactory = new EntityDaoFactory("universe", Settings.Default.ConnectionDb, new TransactionManager()) { ....

Jde o nadstavbu BLToolkitu ktera se postara o to , aby se kod ktery uzavreme do delegata Action , ktery TManageru predame, provedl v jedne transakci . Postara se zaroven o vsechny vlozene transakce a v pripade vyjimky proces rollbackne.

* Na tride DaoSource ted najdeme vlastnost TransactionManager a na ni se daji volat metody.
* Vlozime akce ktere chceme provadet v transakci do delegata Action napr:

Action action = () => { nejake crud operace };

a zavolame :

m\_transactionManager.CallInsideTransaction(action, m\_vlastnostiPlanetDao.CreateCommonDbContext());

Prvnim parametrem je nase akce , a druhym je instance UniverseDataModelu vytvorena pomoci metody CreateCommonDbContext volane na tride m\_vlastnostiPlanetDao(kde je taky explicitne napsana (return CreateDbContext) ).

Metoda CreateDbContext je na tride VlastnostiPlanetDao dostupna, protoze je to metoda rodicovske tridy od ktere dedi. ( EntityMultiKeyDaoBase ) .

**‎15 ‎August, ‎2017**

* Ve tride VlastnostDao jsme prepsali metodu Rodice Insert public override void Insert(Vlastnost obj)

Metodu jsme ale nepouzili , misto toho jsme transakci provedli pomoci TransactionManageru. (DoAllChanges).

# Vesmir

Ve vesmiru jsme pri dokonceni AdoNetu, meli entity bez atributu od niceho nebyly podedene.

Po instalaci balicku BLToolkit, jsme entitam pridali atributy a podedili jsme je od Mitrozovych trid, podle typu primarniho klice.

Napr Galaxie dostala takoveto atributy:

using BLToolkit.DataAccess;

using Mediaresearch.Framework.DataAccess.BLToolkit;

using Mediaresearch.Framework.DataAccess.BLToolkit.DaoFactory;

namespace WpfUniverse.Entities

{

[DaoFactory(DaoType = typeof(GalaxyDao))]

[TableName("Galaxie", Owner = "dbo")]

public class Galaxie : LightDatabaseEntityIdentityIntKey<Galaxie>

{

public string Jmeno { get; set; }

public long PolohaX { get; set; }

public long PolohaY { get; set; }

public long PolohaZ { get; set; }

}

}

GalaxyDao dostalo takoveto podedeni, a dva konstruktory, které bazovce predaji dbAlias

using Mediaresearch.Framework.DataAccess.BLToolkit.Dao;

using System.Data;

namespace WpfUniverse.Entities

{

public class GalaxyDao : EntityIdentityKeyDaoBase<Galaxie, UniverseDataModel, int, Galaxie>, IGalaxyDao

{

public GalaxyDao(string dbAlias)

:base(dbAlias)

{

}

public GalaxyDao(string dbAlias, IsolationLevel isolationLevel)

: base(dbAlias, isolationLevel)

{

}

}

public interface IGalaxyDao : ISimpleDao<Galaxie, int>

{

}

}

Abychom mohli pouzivat BLToolkit potrebujeme DataModel který je podedeny od TransactionManagera. Ma 8 konstruktoru.7

public class UniverseDataModel : TransactionDbManager

public Table<Galaxie> Galaxie => GetTable<Galaxie>();

public Table<Planeta> Planeta => GetTable<Planeta>();

public Table<Vlastnost> Vlastnost => GetTable<Vlastnost>();

public Table<VlastnostiPlanet> VlastnostiPlanet => GetTable<VlastnostiPlanet>();

GetTable<> je metoda na managerovi