IT.InfoTn.Framework

Introduzione

Il framework di accesso ai dati per lo sviluppo rapido di applicazioni può essere utilizzato indipendentemente dalla complessità e dalla dimensione del progetto per evitare l'implementazione di un datalayer specifico.
Le funzionalità offerte in modo trasparente dalla libreria permettono l'accesso a basi dati eterogenee attraverso l'implementazione di dataprovider specifici (l'accesso a database Microsoft Access, SQL Server e Oracle sono già implementati e testati).
Le versioni più recenti permettono l'utilizzo da parte dell'applicazione di sorgenti dati multiple.
Il framework si presta ad essere applicato sia in ambito web che in ambito winforms.

Configurazione di un nuovo progetto

L'architettura del sistema si compone di due blocchi indipendenti: il motore generico che si occupa di offrire un accesso ai dati in modo trasparente e la collezione di oggetti che descrive gli elementi della base dati (tabelle, viste e procedure).
Una nuova applicazione, per poter iniziare, ha quindi bisogno degli assembli del motore compilati e di un insieme di oggetti che descrivono la struttura dei dati.

Definizione struttura e riferimenti

Data types:

Il framework richiede la rimappatura di tutti gli oggetti del database (tabelle, viste e stored procedure) in classi ereditate dai tipi base.
L'operazione di creazione di tali classi viene svolta in modo automatico dall'applicativo IT.InfoTn.Framework.DataTypesCreator che deve essere istruito sulla sorgente dati (Oracle, SQL server e Microsoft Access) e sul namespace del progetto in cui creare le classi.
Il progetto contenente le classi così create deve aggiungere ai propri riferimenti gli assemblies:
IT.InfoTn.Framework.DataTypes
IT.InfoTn.Framework.Interfaces

Le classi create in automatico sono classi partial e possono quindi essere estese liberamente anche per sfruttare alcuni comportamenti ulteriori del framework quali l'uso delle sequences per la generazione degli identificativi in Oracle

Nelle classi generate a partire da tabelle e viste sono esclusi i campi BLOB in quanto oggetto di gestione separata ed i campi CLOB non supportano l'aggiornamento automatico. Tali classi devono essere estese affiancando una partial class che implementi l'interfaccia vuota IFrameworkObjectWithBlob, questo fa sì che l'oggetto possa essere utilizzato per gestire, tramite metodi specifici, i campi BLOB e CLOB nel modo corretto per le diverse tecnologie.

Business layer:

Il punto di accesso al datalayer è rappresentato dalla classe Controller nell'assembly IT.InfoTn.Framework.ClientController.
Il progetto contenente le regole di business dell'applicazione deve aggiungere ai propri riferimenti gli assemblies:
IT.InfoTn.Framework.DataTypes
IT.InfoTn.Framework.Interfaces

IT.InfoTn.Framework.ClientController (nel codice deve essere richiamata l’interfaccia IController, utilizzando un pattern di refactory per caricare l’assembly)

Interfaccia:

Il progetto che implementa l'interfaccia, sia esso un'applicazione Web, un servizio Web, un applicativo Winforms, console o altro deve aggiungere ai propri riferimenti l'assembly IT.InfoTn.Framework.DataStorage (o eventiali implementazioni particolari dell'interfaccia IdataProvider)

Configurazione sorgenti dati

La sorgente dati per l'applicazione deve essere configurata a livello dell'interfaccia.
Possono essere definite da 1 a N sorgenti dati posponendo nel nome dei parametri un identificativo che dovrà essere usato a runtime all'istanziazione di un oggetto Controller. In mancanza di tale parametro il sistema utilizzerà il provider di default che ha come nome parametri DataProviderInitString e DataProviderType.

Esistono due modalità per tale configurazione: tramite file di configurazione oppure tramite registro di Windows.

File di configurazione:

Nel progetto di interfaccia creare un nuovo file di configurazione nominato storage.config contenente la seguente struttura

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
 <appSettings>

  <add key="DataProviderInitString" value="Server=XXX.world;User ID=user;Password=password" />
  <add key="DataProviderType" value="IT.InfoTn.Framework.DataProviders.OracleClientProvider,
IT.InfoTn.Framework.DataProviders"/>

  <add key="DataProviderInitString2" value="Data Source=.\SQLExpress;integrated
security=true;attachdbfilename=[STARTUP]\TestDB.mdf;user instance=true;" />
  <add key="DataProviderType2" value="IT.InfoTn.Framework.DataProviders.SqlClientProvider,
IT.InfoTn.Framework.DataProviders"/>

 </appSettings>
</configuration>
```

Questo metodo richiede l'aggiunta di un ulteriore riferimento per i progetti web all'assembly IT.Infotn.Framework.WebConfig che permette al sistema di reperire a runtime i file di configurazione in quanto il percorso non è predicibile. L'uso del file storage è sconsigliato in tale scenario.

Registry:

Nel file di configurazione del progetto di interfaccia inserire il seguente appsetting

```
<add key="DataSources" value ="XYZ" />
```

E nel registro di windows aggiungere le chiavi relative

```
[HKEY_LOCAL_MACHINE\SOFTWARE\XYZ]

"DataProviderType"="IT.InfoTn.Framework.DataProviders.OracleClientProvider,
IT.InfoTn.Framework.DataProviders.OracleClient"
"DataProviderInitString"="Server=XXX.world;User ID=user;Password=password"

"DataProviderType2"="IT.InfoTn.Framework.DataProviders.SqlClientProvider,
IT.InfoTn.Framework.DataProviders.SQLServer"
"DataProviderInitString2"="Data Source=.\\SQLExpress;integrated
security=true;attachdbfilename=[STARTUP]\\TestDB.mdf;user instance=true;"
```

I due metodi sono equivalenti, scegliere in funzione dell'architettura del progetto, un'architettura SOA con diversi servizi web sulla stessa macchina che accedono alle medesime sorgenti possono trarre vantaggio dall'uso del registro per evitare la gestione di numerosi file di configurazione.

Sorgente dati	ProviderType
Oracle	IT.InfoTn.Framework.DataProviders.OracleClientProvider, IT.InfoTn.Framework.DataProviders.OracleClient
ODP10 (non testato)	IT.InfoTn.Framework.DataProviders.ODPProvider10, IT.InfoTn.Framework.DataProviders.ODP10
Microsoft SQL Server	IT.InfoTn.Framework.DataProviders.SqlClientProvider, IT.InfoTn.Framework.DataProviders.SQLServer
Microsoft Access	IT.InfoTn.Framework.DataProviders.AccessProvider, IT.InfoTn.Framework.DataProviders.Access

ODBC	IT.InfoTn.Framework.DataProviders.OdbcProvider, IT.InfoTn.Framework.DataProviders.Odbc

Nella definizione delle InitString è possibile utilizzare dei segnaposto per percorsi predefiniti nel sistema (da usare in applicazioni windows)

[STARTUP]	Cartella contentente l'applicazione di avvio
[USERAPPDATA]	Cartella contentente i dati applicazioni per l'utente correntemente loggato al sistema (C:\Documents and Settings\<UTENTE>\Dati applicazioni)
[USERDOCUMENTS]	Cartella dei documenti per l'utente correntemente loggato al sistema (C:\Documents and Settings\<UTENTE>\Documenti)
[SHAREDDOCUMENTS]	Cartella dei documenti condividi tra tutti gli utenti del sistema (C:\Documents and Settings\All Users\Documenti)

Filtri sulle date:

La classe SQLFilter che si occupa di generare i filtri SQL in modo guidato tratta le date in formato breve, non considerando quindi nelle dichiarazioni where l'ora. Ciò permette di evitare tutta una serie di problematiche legate alla ricerca di date comprensive dell'ora.

Se in alcuni campi fosse richiesta una ricerca più puntuale anche sull'ora tali campi devono essere enumerati assieme alle informazioni sulla sorgente dati (config o registry) utilizzando la chiave DateTimeFields e come valore la lista dei campi per cui considerare anche l'ora separati dal carattere |.
Es.

```
<add key="DateTimeFields" value="DATAMODIFICA|DATAINSERIMENTO|DATACONVALIDA|TTIMEVAR" />
"DateTimeFields"="DATAMODIFICA|DATAINSERIMENTO|DATACONVALIDA|TTIMEVAR"
```

Se non specificato diversamente tramite tale parametro i campi DATAMODIFICA, DATAINSERIMENTO, DATACONVALIDA e TTIMEVAR saranno automaticamente considerati per l'uso dell'ora nelle ricerche.

Classe Controller

La classe Controller offre l'accesso ai dati attraverso una serie di metodi che possono essere usufruiti dal codice del livello di business attraverso due interfacce:

IController

Gruppo	Metodo	Descrizione
STATUS	System.Data.ConnectionState GetConnectionState()	
	System.Data.ConnectionState GetConnectionStateNoTx()	
	bool TransactionIsClosed { get }	
TRANSACTION	void OpenTransaction()	If the controller object is istantiated using the parameter useTransaction=True the transaction in opened immediatly and is used for every write operation until the closure which is authomatic at the disposal of the controller
	void CloseTransaction(bool commit)	Forces the closure of the transaction (commit or rollback). Automatic commit at the disposal of the controller
CRUD	long InserisciEntita(IFrameworkObject st)	Create a new record with the data of the element
	void ModificaEntita(IFrameworkObject st)	Update the record of the element with the value altered
	void CancellaEntita(IFrameworkObject st, string filter)	Deletes all the entities of the prototype's type which fit the filter
	void CancellaEntita(IFrameworkObject st)	Deletes the entity based on the value of the primary keys set in the object
	T GetElemento<T>(IFrameworkCollection collection, string filter)	Returns the first element fitting the search parameters
	T LeggiElementi<T>(T collection, string filter, string order, ref int risultati)	Reads the elements based on the parameters and the Generic type and returns the elements count
	T LeggiElementi<T>(T collection, string filter, string order)	Reads the elements based on the parameters and the Generic type
	T LeggiElementiPaginati<T>(T collection, string filter, string order, int pagina, int elementiPagina, ref int risultati)	Reads the elements based on the parameters and the Generic type and return the elements count
	string[] GetItemsField<T>(T collection, string filtro, string field)	Reads the elements based on the parameters and the Generic type and returns an array containing the DISTINCT values of the specified field for each element found
	T LeggiElementi<T, K>(T collection, string filter, string order, K searchCollection, string field, string searchField)	Reads the elements which are in the result set of another search
	T LeggiElementiPaginati<T, K>(T collection, string filter, string order, K searchCollection, string field, string searchField, int pagina, int elementiPagina, ref int risultati)	Reads the elements paginated which are in the result set of another search
	void LaunchStoredProcedure(IFrameworkStoredProcedure proc)	Executes the Function or Stored Procedure and eventually fills the output params
INVESTIGATION	int ConteggioElementi(IFrameworkCollection collection, string filter)	Counts the elements which fit the filter
	string GetSQLCommand(IFrameworkCollection collection, string filter, string order, int pagina, int elementiPagina)	Returns the SQL text command composed starting from the parameters
	string GetSQLCommand(IFrameworkCollection collection, string filter, string order)	Returns the SQL text command composed starting from the parameters
BLOB	byte[] ReadBlob(IFrameworkObjectWithBlob item, string blobField)	Return the binary content of the BLOB field
	void WriteBlob(IFrameworkObjectWithBlob item, string blobField, byte[] blobData)	Writes the binary in the BLOB field
	void WriteClob(IFrameworkObjectWithBlob item, string clobField, string clobData)	Writes the binary in the CLOB field

IControllerSQLText

L’interfaccia permette di superare gli automatismi offerti dal framework per quelle situazioni in cui ciò si rende necessario. Non va usata se non indispensabile.

Gruppo	Metodo	Descrizione
INVESTIGATION	int ConteggioElementi(string comandoSQL)	Counts the elements based on the SQL command
CRUD	T LeggiElementiSQL<T>(T collection, string comandoSQL)	Reads the elements based on the SQL command and the Generic type
	T LeggiElementiSQL<T>(T collection, string comandoSQL, ref int risultati)	Reads the elements based on the SQL command and the Generic type and returns the elements count
	T LeggiElementiSQLPaginati<T>(T collection, string comandoSQL, int pagina, int elementiPagina, ref int risultati)	Reads the elements based on the SQL command and the Generic type with pagination and returns the totale elements count

Classe SQLFilter

Sql filter permette di comporre un filtro sql pezzo per pezzo in modo “intelligente”.

Congiunzioni logiche:

L’uso di AND o di OR per congiungere i diversi parametri del filtro è automatico:

1. se la classe è istanziata con il parametro FiltroInOr a true tutti i filtri sono in OR
2. altrimenti in caso di campo/key uguale i filtri sono in OR, altrimenti sono in AND

Composizione:

Il filtro va composto utilizzando il metodo ADD della classe nella diverse forme:

```
Add(string key, string filter)
Aggiunta di un filtro SQL
  · chiave per gestione operatori logici
  · filtro SQL da applicare

Add(string campo, OperatoriFiltro operatore, object valore)
Aggiunta di un filtro su di un campo
  · campo da filtrare
  · operatore di filtro
  · valore o SQL nidificato per cui filtrare

Add(string campo, OperatoriFiltro operatore, object valore, string distintivo)
Aggiunta di un filtro su di un campo (in caso di AND sullo stesso campo)
  · campo da filtrare
  · operatore di filtro
  · valore o SQL nidificato per cui filtrare
  · suffisso distintivo del campo per oerare AND al posto di OR

Add(string campo, OperatoriFiltroAvanzati operatore, object valore, int score)
Aggiunta di un filtro di tipo a avanzato (es. Fuzzy contains)
  · campo da filtrare
  · operatore di filtro
  · valore per cui filtrare
  · ampiezza del filtro (valore da 1 a 80)

Add(string campo, OperatoriFiltroMultipli operatore, object valore1, object valore2)
Aggiunta di un filtro di tipo a valori multipli (es. Between)
  · campo da filtrare
  · operatore di filtro
  · valore per cui filtrare
  · valore per cui filtrare
```

Operatori filtro:

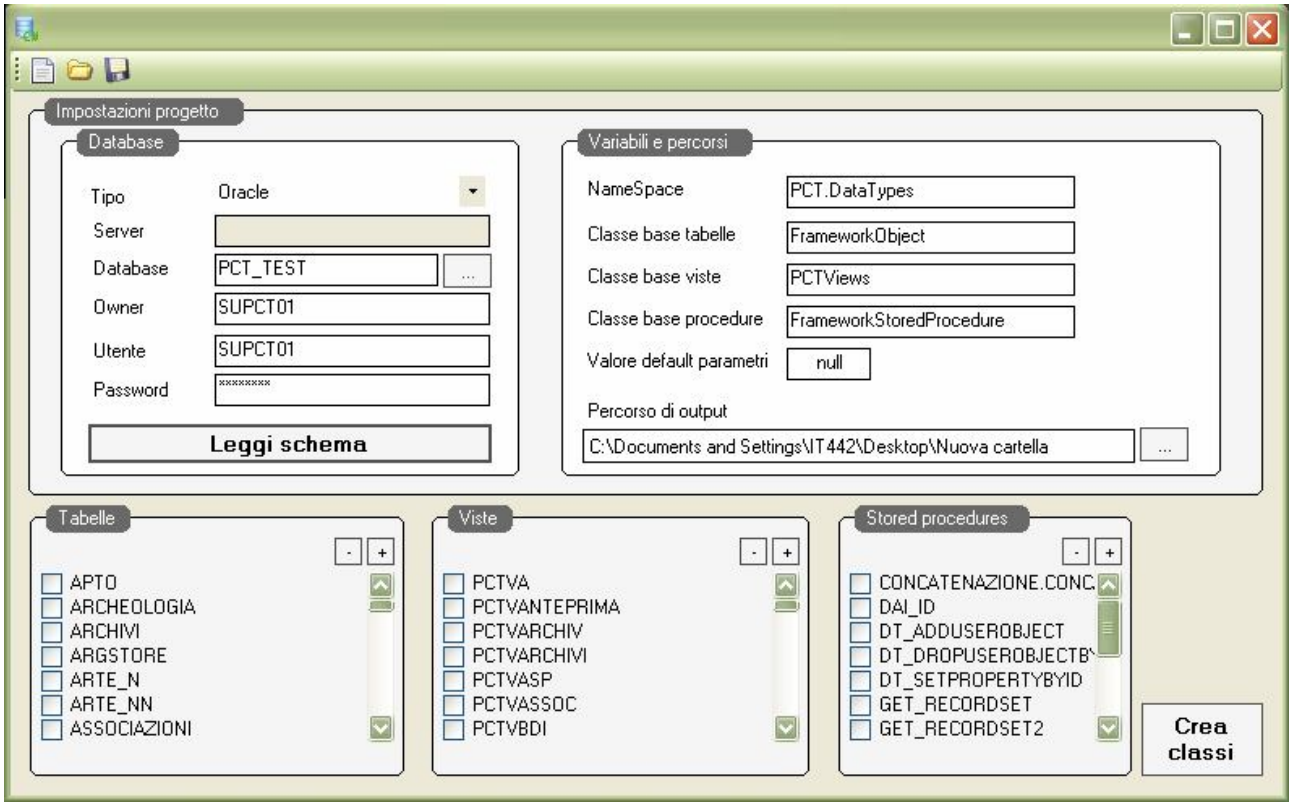
```
OperatoriFiltro.Uguale:                [CAMPO] = '[VALORE]'
OperatoriFiltro.Diverso:               [CAMPO] <> '[VALORE]'
OperatoriFiltro.Maggiore:              [CAMPO] > '[VALORE]'
OperatoriFiltro.Minore:                [CAMPO] < '[VALORE]'
OperatoriFiltro.MaggioreUguale:        [CAMPO] >= '[VALORE]'
OperatoriFiltro.MinoreUguale:          [CAMPO] <= '[VALORE]'
OperatoriFiltro.Contiene:              UPPER([CAMPO]) LIKE UPPER('%[VALORE]%')
OperatoriFiltro.IniziaCon:             UPPER([CAMPO]) LIKE UPPER('[VALORE]%')
OperatoriFiltro.FinisceCon:            UPPER([CAMPO]) LIKE UPPER('%[VALORE]')
OperatoriFiltro.Simile:                UPPER([CAMPO]) LIKE UPPER('[VALORE]')
OperatoriFiltro.In:                    [CAMPO] IN([VALORE])
OperatoriFiltro.NotIn:                 [CAMPO] NOT IN([VALORE])
OperatoriFiltro.Contains:              CONTAINS([CAMPO], '[VALORE]', 1)>0 "

OperatoriFiltroMultipli.Between:       ([CAMPO] BETWEEN [VALORE1] AND [VALORE2])

OperatoriFiltroAvanzati.FuzzyContain:  CONTAINS([CAMPO], 'fuzzy([VALORE],[SCORE],50,weight)', 1)>0
```

Data Types

Gli oggetti di database che vengono rimappati nelle classi specifiche dal generatore sono Tabella, Viste e Procedure.
Sono supportate origini dati Oracle, SQL Server, SQL Server file (.mdf) e Microsoft Access. Una volta impostati i parametri di connessione e le informazioni per la generazione dei file è sufficiente selezionare gli oggetti da generare dalle liste popolate interrogando le sorgenti dati e avviare il processo di generazione.



Le configurazioni del generatore per un progetto possono essere salvate per un uso successivo in file specifici associati all'applicazione (.fwk).

Tabelle:

```
namespace NAMESPACE.TABELLE
{
    public partial class NOMETABELLA : CLASSE_BASE_TABELLE

    public class NOMETABELLACollection : List<NOMETABELLA>, IFrameworkCollection
}
```

La classe partial contiene come property i campi della tabella marcate con l'attributo [MappedField()] che viene utilizzato nella costruzione dei comandi SQL per distinguere gli effettivi campi della tabella da eventuali altre proprietà aggiunte in estensioni della classe. I campi chiave sono marcati con l'attributo [PrimaryKey()], tale attributo è essenziale al funzionamento del sistema e ne va verificata la presenza nelle classi generate.
L'attributo [MaxLength(X)] è invece assegnato ai campi di testo dove X contiene il numero massimo di caratteri specificato nel database. Quando viene richiesto l'inserimento e la modifica di un record tale informazione viene verificata e se non rispettata viene lanciata una eccezione.
L'ultimo attributo applicabile è [NullableForeignKey()] che determina l'inserimento del valore Null su di un campo Foreign Key in presenza di valori numerici negativi o stringa vuota.

La classe contiene una serie di proprietà di Self-knowledge che indicano la tabella di riferimento, i momi dei campi da usare per la generazione dei filtri, … e la gestione della serializzazione/deserializzazione.

Per ogni tabella viene anche creata la relativa classe Collection che viene utilizzata nelle chiamate al controller.

Una delle informazioni da inserire in una classe di estensione è l'eventuale sequence da utilizzare per la generazione degli ID dei record in fase di creazione. Se l'architettura dovesse prevedere una sequence unica per tutto il database il modo migliore consiste nel creare una classe base che erediti da FrameworkObject in cui effettuare l'override della property public virtual string SequenceName() e generare le classi indicando come base per le tabella la nuova classe creata.

È possibile iniettare nella select ulteriori informazioni implementando l'interfaccia IFrameworkObjectExtension che permette l'indicazione degli ulteriori campi da estrarre tramite la property string AggiunteSelect().

Viste:

La struttura descrittiva delle viste rispecchia quella delle tabelle a parte la decorazione delle proprietà con gli attributi in quanto non è consentito utilizzare un oggetto di tipo vista per le operazioni di scrittura dati.

Procedure:

Le stored procedure e le function sia di Oracle che di SQL server sono rappresentate da una collezione dei parametri in ingresso e uscita e possono essere lanciate attraverso la specifica funzione della classe Controller che si occupa di riempire con gli eventuali valori in uscita i parametri e di gestire in modo automatico i campi BLOB e i campi CLOB/MEMO.