DYNACRUD

Gestore CRUD dinamiche (ma non solo)

by Fabrizio Ricciarelli per Eustema SpA

07-12-2015

Documento tecnico

1. Scopo

Obiettivo del presente documento è quello di esporre le funzionalità, a livello macroscopico, di un prodotto orientato principalmente ad un'audience di sviluppatori software operanti con tecnologie Microsoft Sql Server + C# .Net Framework 3.5 – 4.5

DYNACRUD (contrazione dei termini "Dynamic" e "CRUD", quest'ultimo acronimo di "Create, Read, Update and Delete – quindi CRUD dinamica) può essere catalogato come strumento RAD (Rapid Application Development): il suo impiego è estremamente flessibile in quanto il prodotto può essere configurato in ogni sua minima parte essendo esso composto, principalmente, da moduli di scripting T-SQL (essenzialmente funzioni – Table Valued Functions e Scalar Valued Functions). Questo strumento si colloca principalmente sullo strato dati (Data Access Layer), quindi sul back-end ma ciò che riesce a produrre dinamicamente ha delle ripercussioni immediate sullo strato intermedio di business – che viene letteralmente scritto in codice nativo C# (o qualunque altro linguaggio, se opportunamente adattato) dallo strumento - , pronto per essere utilizzato, attraverso i suoi connettori/interfacce, con strumenti analoghi ma più orientati allo strato di presentation.

2. Campi di applicazione

Gli ambiti e gli scenari nei quali il prodotto DYNACRUD è più idoneo ad essere impiegato sono i seguenti:

- sviluppo di web applications dove sono richiesti molteplici moduli di input/output (data-entry e preparazione alla reportistica di massa)
- sviluppo di interfacce three-tier laddove necessitino numerose operazioni di lettura/scrittura/modifica sulle basi di dati, anche eterogenee (ad es. SQL <=> DB2)
- sviluppo di batches/console applications per migrazioni dati in background

3. Vantaggi

Ciò che rende vantaggioso l'uso di un sistema RAD è principalmente la riduzione drastica dei tempi di sviluppo e manutenzione dei moduli di interfaccia da/verso le basi di dati: DYNACRUD si occupa di interpretare la struttura di una o più tabelle (preventivamente progettate al meglio, in terza forma normale, secondo strutture logiche che rispettino i più rigorosi canoni richiesti dai RDBMS); ma non solo. Anche modelli meno organizzati e strutturati potranno, con gli opportuni accorgimenti, essere recepiti da DYNACRUD e trasformati in blocchi di codice perfettamente funzionanti nei modelli three-tier (Presentation, Business, Data layer) o nei moduli batch.

4. Cosa fa DYNACRUD

Questo sistema RAD svolge, in tempi molto rapidi, essenzialmente questi compiti:

- legge la definizione di una qualsiasi tabella SQL (ad oggi è DYNACRUD è stato progettato e sviluppato per interfacciarsi con tabelle realizzate in ambiente Microsoft Sql Server 2012 ma è retrocompatibile fino alla versione 2008; non è difficile, grazie alla sua flessibilità, adattarlo ad altri ambienti o ad altre versioni) o quella di una vista (che mette, ad esempio, in collegamento più tabelle contemporaneamente)
- espone, arricchendole, le proprietà intrinseche delle colonne della tabella/vista, operando opportuni mappings con altri ambienti (ad esempio, trasforma il tipo di dato di una determinata colonna, nel suo equivalente in linguaggio C#) nonché fornendo dati di popolamento casuali (per le fasi di test) aderenti al tipo di dato della colonna in corso di arricchimento
- fornisce un meccanismo di templating (ovvero di modellazione basata su impianti draft + placeholders) tramite il quale si può disegnare un modello, definire dove andranno posti i valori contenuti nelle colonne del database e vedere questi popolati automaticamente da DYNACRUD
- produce, sempre avvalendosi del templating, un numero pressoché infinito di elenchi basati sulle colonne cui si aggancia: tra questi viene data maggior evidenza ai GridView in C#
- mette a disposizione una nutrita libreria di funzioni T-SQL svolgenti numerosi compiti ripetitivi senza che sia necessario svilupparne altre ad-hoc
- introduce un sistema di source repository (archivio di moduli sorgenti) che viene sfruttato per gli arricchimenti e prepara la strada, in una versione futura, al CodeDom, ovvero alla possibilità di eseguire “chunk”, ossia porzioni di codice sorgente, direttamente a runtime con compilazione ed esecuzione istantanea in memoria (senza la necessità, quindi, di intraprendere nuovamente tutta la fase di deploy); tramite questo sistema è possibile aggiungere funzionalità alle classi C# generate, con metodi pubblici e/o privati secondo specifiche necessità
- produce classi C# (ma anche Java o VB) che definiscono e gestiscono oggetti rappresentanti la tabella/vista dalla quale sono stati generati
- produce codice ASPX, e relativo code behind, per la gestione completa (tutti gli eventi) di un GridView dinamico secondo la sintassi del .NET framework 3.5
- produce codice T-SQL dinamico per eventuali manipolazioni alla struttura tabellare

5. Elenco delle funzionalità

DYNACRUD mette a disposizione i seguenti oggetti:

Tipologia	Denominazione	Funzione
Tabella	FILLER_DEC	Contenitore dei placeholders (segnaposto) per il templating (modellazione)
Tabella	SourcesRepository	Tabella preposta a contenere il codice sorgente (completo o parziale), in vari linguaggi, per la creazione degli elenchi di definizione degli oggetti per la crud dinamica
Tabella	RepositoryTypes	Contenitore delle tipologie di codice sorgente
Vista	V_SourcesRepository	Vista di raccordo tra le tabelle *SourcesRepository* e *RepositoryTypes*
Vista	V_RAND	Vista per estrazione di un numero casuale
Vista	V_NEWID	Vista per estrazione di un GUID casuale
Vista	V_RAND_NEWID	Vista preposta al ritorno di - un valore reale (float, double) casuale - un GUID casuale - un BIT casuale
TVF (Table-Valued Function)	fnGetTableDef	Funzione preposta alla rappresentazione della struttura, arricchita del mapping con C# e di dati casuali di esempio, di una tabella SQL
TVF	fnGetSpDef	Funzione preposta alla rappresentazione della struttura di una SP SQL
SVF (Scalar Valued Function)	fnFillSqlTemplateWithFieldsList	Funzione preposta al riempimento di un template sostituendo le wildcards ($N,$T,$F, etc.) con i rispettivi rimpiazzi provenienti dalla funzione dbo.fnGetTableDef
SVF (Scalar Valued Function)	fnFillSqlTemplateWithFieldsListOrdered	Idem c.s. ma con la specifica del tipo di dato secondo il quale operare l'ordinamento
SVF	fnCleanVariableName	Funzione preposta al rimpiazzo di stringhe utilizzate come nomi di variabli
SVF	fnCamelCaseUnderscoredTableName	Funzione preposta alla trasformazione di una stringa nella sua versione "Camel Case" (i caratteri di underscore "_" divengono lettere iniziali maiuscole. Ad esempio, la stringa "ENTITA_DETT" diventa "EntitaDett"
SVF	fnBuildCsharpClass	Funzione preposta alla creazione di una classe C# partendo dalla struttura di una tabella SQL
SVF	fnRandomString	Funzione che ritorna una stringa casuale di lunghezza specifica
SVF	fnGetRandomDataByDataType	Funzione che ritorna una stringa casuale di lunghezza specifica in base al tipo di dato specificato. Se indicato, puo' racchiudere il dato casuale tra delimitatori
SVF	fnAddSetParam	Funzione preposta alla elencazione di un insieme di "set" di nomi di campo per la creazione di uno statement di update
SVF	fnGetCsharpSqlMapperClass	Funzione preposta alla creazione della classe C# statica 'SqlMapper' la quale provvede a fornire i metodi per la conversione da SqlDbType a Type e viceversa
SVF	fnBuildCsharpGridViewASPX	Funzione preposta alla creazione della definizione degli elementi ASPX per un GridView partendo dalla struttura di una tabella SQL
SVF	fnBuildCsharpGridViewCS	Funzione preposta alla creazione del codice C# del CodeBehind di gestione degli eventi di un GridView partendo dalla struttura di una tabella SQL
SVF	fnBuildCsharpGridViewCSS	Funzione preposta alla creazione della definizione degli elementi CSS per un GridView partendo dalla struttura di una tabella SQL
SVF	fnGetSource	Funzione preposta al recupero di un elemento dal repository dei sorgenti
SVF	fnBuildStoredProcedureForCRUD	Funzione preposta alla creazione della Stored Procedure per la gestione di una CRUD dinamica partendo dalla struttura di una tabella

		SQL
Stored Procedure	spSelInsUpdDelEntitaDett	Stored procedure per CRUD dinamica (simil-entity framework) Select,Insert,Update,Delete su tabella ENTITA_DETT orientata alla costruzione del codice sql dinamico ad incapsulamento annidato progressivo

Nel presente documento, a titolo di esempio, verrà utilizzata una tabella "ENTITA_DETT" per rappresentare il risultato dell'invocazione delle varie funzioni SQL di DYNACRUD.

La struttura SQL della tabella "ENTITA_DETT" è la seguente:

```
CREATE TABLE dbo.ENTITA_DETT
(
   CodiceEntita varchar(5) NOT NULL,
      CFCreditore varchar(16) NOT NULL,
      CFDebitore varchar(16) NULL,
      CodicePrestazione varchar(5) NULL,
      CodiceSede char(6) NOT NULL,
      CodiceProcedura varchar(4) NOT NULL,
      Progressivo int NOT NULL,
      Anno int NOT NULL,
      Mese varchar(2) NOT NULL,
      AnnoRif int NULL,
      MeseRif varchar(2) NULL,
      ImportoCredito decimal(18, 2) NULL,
      ImportoDebito decimal(18, 2) NULL,
      ImportoSospeso decimal(18, 2) NULL,
      ImportoSospesoInAtto decimal(18, 2) NULL,
      DataInserimento datetime NOT NULL,
      DataUltimaModifica datetime NOT NULL,
      CodiceRegione varchar(2) NOT NULL,
      IdStruttura int NOT NULL,
      ChiaveARCAAnagraficaCodice varchar(3) NULL,
      ChiaveARCAAnagraficaProgressivo int NULL,
      ChiaveARCAPrestazione varchar(128) NULL,
      CONSTRAINT PK_ENTITA_DETT PRIMARY KEY CLUSTERED
      (
            CodiceEntita ASC,
            CFCreditore ASC,
            CodiceSede ASC,
            CodiceProcedura ASC,
            Progressivo ASC,
            Anno ASC,
            Mese ASC,
            IdStruttura ASC
      )
      WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
      ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
```

In verde sono state evidenziate le colonne "nullabili", mentre in rosso le colonne chiave (primary key); questa differenziazione, come indicato nel paragrafo successivo, viene marcata da DYNACRUD attraverso degli indicatori in forma di suffissi ai nomi dei campi, allo scopo di semplificare l'identificazione di questi tipi di colonne per lo sviluppatore.

Negli esempi che seguono sarà possibile comprendere come questi identificatori vengono implementati e come utilizzarli.

6. Esempi di invocazione delle funzioni e relativi risultati

6.1. Funzione **fnGetTableDef** (Table-Valued function)

```
SELECT * FROM dbo.fnGetTableDef('ENTITA_DETT')
```

fieldName	variableName	castedFieldName	castedDenulledFieldName	fieldType	fullFieldType	SqlDbType	cSharpType	cSharpPrivateVariableName	cSharpPublicPropertyName	fieldLength	stringFieldLength	fieldIsNullable	fieldIsIdentity	fieldIsKey	fieldPrecision	fieldScale	randomData
[CodiceEntita]	@CodiceEntita	@CodiceEntita	ISNULL(@CodiceEntita,'')	varchar	varchar(5)	SqlDbType.VarChar	String	_codiceEntita_PK	CodiceEntita_PK	5	5	0	0	1	0	0	"57PMB"
[CFCreditore]	@CFCreditore	@CFCreditore	ISNULL(@CFCreditore,'')	varchar	varchar(16)	SqlDbType.VarChar	String	_cFCreditore_PK	CFCreditore_PK	16	16	0	0	1	0	0	"FZNITUBXKAEDFX4F"
[CFDebitore]	@CFDebitore	@CFDebitore	@CFDebitore	varchar	varchar(16)	SqlDbType.VarChar	String	_cFDebitore_x	CFDebitore_x	16	16	1	0	0	0	0	"WDLWMI5ZCUFRKJUI"
[CodicePrestazione]	@CodicePrestazione	@CodicePrestazione	@CodicePrestazione	varchar	varchar(5)	SqlDbType.VarChar	String	_codicePrestazione_x	CodicePrestazione_x	5	5	1	0	0	0	0	"7BM"
[CodiceSede]	@CodiceSede	@CodiceSede	ISNULL(@CodiceSede,'')	char	char(6)	SqlDbType.Char	String	_codiceSede_PK	CodiceSede_PK	6	6	0	0	1	0	0	"6EJRWR"
[CodiceProcedura]	@CodiceProcedura	@CodiceProcedura	ISNULL(@CodiceProcedura,'')	varchar	varchar(4)	SqlDbType.VarChar	String	_codiceProcedura_PK	CodiceProcedura_PK	4	4	0	0	1	0	0	"ACID"
[Progressivo]	@Progressivo	CAST(@Progressivo AS varchar(10))	ISNULL(CAST(@Progressivo AS varchar(10)), '')	int	int	SqlDbType.Int	int	_progressivo_PK	Progressivo_PK	4	10	0	0	1	10	0	6909
[Anno]	@Anno	CAST(@Anno AS varchar(10))	ISNULL(CAST(@Anno AS varchar(10)), '')	int	int	SqlDbType.Int	int	_anno_PK	Anno_PK	4	10	0	0	1	10	0	6216
[Mese]	@Mese	@Mese	ISNULL(@Mese,'')	varchar	varchar(2)	SqlDbType.VarChar	String	_mese_PK	Mese_PK	2	2	0	0	1	0	0	"CR"
[AnnoRif]	@AnnoRif	CAST(@AnnoRif AS varchar(10))	CAST(@AnnoRif AS varchar(10))	int	int	SqlDbType.Int	int?	_annoRif_x	AnnoRif_x	4	10	1	0	0	10	0	7956
[MeseRif]	@MeseRif	@MeseRif	@MeseRif	varchar	varchar(2)	SqlDbType.VarChar	String	_meseRif_x	MeseRif_x	2	2	1	0	0	0	0	"S4"
[ImportoCredito]	@ImportoCredito	CAST(@ImportoCredito AS varchar(20))	CAST(@ImportoCredito AS varchar(20))	decimal	decimal(18,2)	SqlDbType.Decimal	Decimal?	_importoCredito_x	ImportoCredito_x	9	20	1	0	0	18	2	436735.49
[ImportoDebito]	@ImportoDebito	CAST(@ImportoDebito AS varchar(20))	CAST(@ImportoDebito AS varchar(20))	decimal	decimal(18,2)	SqlDbType.Decimal	Decimal?	_importoDebito_x	ImportoDebito_x	9	20	1	0	0	18	2	119073.64
[ImportoSospeso]	@ImportoSospeso	CAST(@ImportoSospeso AS varchar(20))	CAST(@ImportoSospeso AS varchar(20))	decimal	decimal(18,2)	SqlDbType.Decimal	Decimal?	_importoSospeso_x	ImportoSospeso_x	9	20	1	0	0	18	2	358438.99
[ImportoSospesoInAtto]	@ImportoSospesoInAtto	CAST(@ImportoSospesoInAtto AS varchar(20))	CAST(@ImportoSospesoInAtto AS varchar(20))	decimal	decimal(18,2)	SqlDbType.Decimal	Decimal?	_importoSospesoInAtto_x	ImportoSospesoInAtto_x	9	20	1	0	0	18	2	992729.81
[DataInserimento]	@DataInserimento	CONVERT(varchar(26),@DataInserimento,120)	ISNULL(CONVERT(varchar(26),@DataInserimento,120),GETDATE())	datetime	datetime	SqlDbType.DateTime	DateTime	_dataInserimento	DataInserimento	8	26	0	0	0	23	3	"2017-12-26 10:35:18.400"
[DataUltimaModifica]	@DataUltimaModifica	CONVERT(varchar(26),@DataUltimaModifica,120)	ISNULL(CONVERT(varchar(26),@DataUltimaModifica,120),GETDATE())	datetime	datetime	SqlDbType.DateTime	DateTime	_dataUltimaModifica	DataUltimaModifica	8	26	0	0	0	23	3	"2018-08-04 10:35:18.400"
[CodiceRegione]	@CodiceRegione	@CodiceRegione	ISNULL(@CodiceRegione,'')	varchar	varchar(2)	SqlDbType.VarChar	String	_codiceRegione	CodiceRegione	2	2	0	0	0	0	0	"J6"
[IdStruttura]	@IdStruttura	CAST(@IdStruttura AS varchar(10))	ISNULL(CAST(@IdStruttura AS varchar(10)), '')	int	int	SqlDbType.Int	int	_idStruttura_PK	IdStruttura_PK	4	10	0	0	1	10	0	6604

La funzione fnGetTableDef opera, principalmente, un "mapping" tra i tipi di dati SQL e quelli C# ma non soltanto: come è possibile vedere all'interno della precedente tabella, vengono anche creati:

- dei nomi di variabili squisitamente nativi SQL (preceduti dalla chiocciola "@"),
- dei cast (trasformazione del tipo di dato per esigenze rappresentative),
- le "denullazioni" tipiche SQL (ISNULL(nomecampo,valoreDiDefaultSeNull)) per esigenze gestionali,
- dati casuali di esempio (cambiano ad ogni invocazione della presente funzione) per esigenze di test e debug

La prima colonna, "fieldName", contiene il nome del campo della tabella fisica racchiuso tra parentesi quadre (square brackets) poiché, in altre possibili circostanze, il nome di campo di una qualsiasi tabella fisica potrebbe prevedere uno spazio tra due termini (ad es.: [Anno Riferimento]) quindi, qualora ciò accadesse, DYNACRUD "impone" questa forma per i nomi di colonna allo scopo di evitare errori di localizzazione di nomi di campo in presenza di spazi.

Come si potrà notare, ai nomi dei campi di tipo "nullable" - nella tabella fisica - vengono aggiunti, nelle colonne "cSharpPrivateVariableName" e "cSharpPublicPropertyName", i suffissi "_x" allo scopo di agevolare, come poc'anzi detto, lo sviluppatore nell'individuazione di tali tipi di campo (per alcuni tipi di dato, quali ad esempio interi, decimali, date, booleani etc., vi è evidenza della "nullabilità" in quanto il linguaggio C# prevede l'identificatore "?" – ad esempio "int? nomevariabile" – ma nei tipi di dato stringa, ed altri, il linguaggio non prevede l'impiego di tale identificatore).

Allo stesso modo, per i tipi di colonna chiave (Primary Key, PK) viene aggiunto, sempre ai nomi di variabile "cSharpPrivateVariableName" e "cSharpPublicPropertyName", il suffisso "_PK".

E' dunque facile intuire che questa funzione è alla base di tutte le altre in quanto, partendo da una serie di definizioni così nutrita, sono state sviluppate le altre funzioni derivate che sfruttano ogni singola colonna ritornata dalla funzione fnGetTableDef.

6.2. Funzione **fnGetSpDef** (Table-Valued function)

```
SELECT * FROM dbo.fnGetSpDef('spSelInsUpdDelEntitaDett')
```

paramName	paramType	fullparamType	SqlDbType	cSharpType	cSharpPrivateVariableName	cSharpPublicPropertyName	paramLength	stringparamLength	paramPrecision	paramScale
@CodiceEntita	varchar	varchar(5)	SqlDbType.VarChar	String	_codiceEntita	CodiceEntita	5	5	5	NULL
@CFCreditore	varchar	varchar(16)	SqlDbType.VarChar	String	_cFCreditore	CFCreditore	16	16	16	NULL
@CFDebitore	varchar	varchar(16)	SqlDbType.VarChar	String	_cFDebitore	CFDebitore	16	16	16	NULL
@CodicePrestazione	varchar	varchar(5)	SqlDbType.VarChar	String	_codicePrestazione	CodicePrestazione	5	5	5	NULL
@CodiceSede	char	char(6)	SqlDbType.Char	String	_codiceSede	CodiceSede	6	6	6	NULL
@CodiceProcedura	varchar	varchar(4)	SqlDbType.VarChar	String	_codiceProcedura	CodiceProcedura	4	4	4	NULL
@Progressivo	int	int	SqlDbType.Int	int	_progressivo	Progressivo	4	10	10	0
@Anno	int	int	SqlDbType.Int	int	_anno	Anno	4	10	10	0
@Mese	varchar	varchar(2)	SqlDbType.VarChar	String	_mese	Mese	2	2	2	NULL
@AnnoRif	int	int	SqlDbType.Int	int	_annoRif	AnnoRif	4	10	10	0
@MeseRif	varchar	varchar(2)	SqlDbType.VarChar	String	_meseRif	MeseRif	2	2	2	NULL
@ImportoCredito	decimal	decimal(18,2)	SqlDbType.Decimal	Decimal	_importoCredito	ImportoCredito	9	20	18	2
@ImportoDebito	decimal	decimal(18,2)	SqlDbType.Decimal	Decimal	_importoDebito	ImportoDebito	9	20	18	2
@ImportoSospeso	decimal	decimal(18,2)	SqlDbType.Decimal	Decimal	_importoSospeso	ImportoSospeso	9	20	18	2
@ImportoSospesoInAtto	decimal	decimal(18,2)	SqlDbType.Decimal	Decimal	_importoSospesoInAtto	ImportoSospesoInAtto	9	20	18	2

La struttura della definizione ritornata dalla funzione fnGetSpDef è pressoché sovrapponibile a quella della sorella maggiore *fnGetTableDef*, ad eccezione del fatto che – stavolta – ad esser presa in analisi è una Stored Procedure piuttosto che una tabella; chiaramente, trattandosi di parametri, piuttosto che di campi, ad essere analizzati, molte caratteristiche/tipologie di mapping non sono qui presenti.

Questa funzione, a differenza di tutte le altre, non ha relazioni di dipendenza con il parco funzioni della libreria DYNACRUD ma è stata in essa inserita a scopo utilitaristico per lo sviluppatore.

6.3. Funzione **fnFillSqlTemplateWithFieldsList** (Scalar-Valued function)

```
PRINT (dbo.fnFillSqlTemplateWithFieldsList('$N, -- (ad es.: $?)' + CHAR(13),'ENTITA_DETT'))
[CodiceEntita], -- (ad es.: "CFMJ2")
[CFCreditore], -- (ad es.: "BWKC6GTEJ3ZWJEWG")
[CFDebitore], -- (ad es.: "WZXSBX3XJPRHACBS")
[CodicePrestazione], -- (ad es.: "IKFHC")
[CodiceSede], -- (ad es.: "RTDUOZ")
[CodiceProcedura], -- (ad es.: "UDVT")
[Progressivo], -- (ad es.: 2966)
[Anno], -- (ad es.: 6078)
[Mese], -- (ad es.: "BC")
[AnnoRif], -- (ad es.: 7742)
[MeseRif], -- (ad es.: "PG")
[ImportoCredito], -- (ad es.: 887126.48)
[ImportoDebito], -- (ad es.: 152372.44)
[ImportoSospeso], -- (ad es.: 356524.92)
[ImportoSospesoInAtto], -- (ad es.: 383326.21)
[DataInserimento], -- (ad es.: "2017-11-06 12:19:54.420")
[DataUltimaModifica], -- (ad es.: "2017-10-11 12:19:54.420")
[CodiceRegione], -- (ad es.: "QS")
[IdStruttura], -- (ad es.: 4668)
[ChiaveARCAAnagraficaCodice], -- (ad es.: "T5Z")
[ChiaveARCAAnagraficaProgressivo], -- (ad es.: 1891)
[ChiaveARCAPrestazione], -- (ad es.:
"SCULXABGHMSTTZGRSVMK2FYYEXYFO4VJBTEFKJZGJKB6E6UMYTCDPKXQKAEXDG54F3BJ4QBHXDDT4RXHEFAC4HXTNJMBFCMGBBEPFGC
D4N2GHFCUSPGZGPT5WDBP6")
```

```
PRINT (dbo.fnFillSqlTemplateWithFieldsList('AND ($N = $V OR $V IS NULL)' + CHAR(13),'ENTITA_DETT'))
AND ([CodiceEntita] = @CodiceEntita OR @CodiceEntita IS NULL)
AND ([CFCreditore] = @CFCreditore OR @CFCreditore IS NULL)
AND ([CFDebitore] = @CFDebitore OR @CFDebitore IS NULL)
AND ([CodicePrestazione] = @CodicePrestazione OR @CodicePrestazione IS NULL)
AND ([CodiceSede] = @CodiceSede OR @CodiceSede IS NULL)
AND ([CodiceProcedura] = @CodiceProcedura OR @CodiceProcedura IS NULL)
AND ([Progressivo] = @Progressivo OR @Progressivo IS NULL)
AND ([Anno] = @Anno OR @Anno IS NULL)
AND ([Mese] = @Mese OR @Mese IS NULL)
AND ([AnnoRif] = @AnnoRif OR @AnnoRif IS NULL)
AND ([MeseRif] = @MeseRif OR @MeseRif IS NULL)
AND ([ImportoCredito] = @ImportoCredito OR @ImportoCredito IS NULL)
AND ([ImportoDebito] = @ImportoDebito OR @ImportoDebito IS NULL)
AND ([ImportoSospeso] = @ImportoSospeso OR @ImportoSospeso IS NULL)
AND ([ImportoSospesoInAtto] = @ImportoSospesoInAtto OR @ImportoSospesoInAtto IS NULL)
AND ([DataInserimento] = @DataInserimento OR @DataInserimento IS NULL)
AND ([DataUltimaModifica] = @DataUltimaModifica OR @DataUltimaModifica IS NULL)
AND ([CodiceRegione] = @CodiceRegione OR @CodiceRegione IS NULL)
AND ([IdStruttura] = @IdStruttura OR @IdStruttura IS NULL)
AND ([ChiaveARCAAnagraficaCodice] = @ChiaveARCAAnagraficaCodice OR @ChiaveARCAAnagraficaCodice IS NULL)
AND ([ChiaveARCAAnagraficaProgressivo] = @ChiaveARCAAnagraficaProgressivo OR @ChiaveARCAAnagraficaProgressivo IS NULL)
AND ([ChiaveARCAPrestazione] = @ChiaveARCAPrestazione OR @ChiaveARCAPrestazione IS NULL)
```

Da questi due semplici esempi è possibile comprendere le infinite possibilità offerte dalla tecnica detta di "templating" (modellazione): dato un modello di base costante, vi si collocano all'interno dei "PlaceHolders", "WildCards" o segnaposto, che saranno rimpiazzati dai corrispondenti elementi definiti dagli identificatori e replicati per ciascun nome di campo presente nella tabella specificata (ENTITA_DETT in questo caso). I segnaposto, riconoscibili dal prefisso "$" e seguiti da una specifica

lettera, corrispondono alle colonne ritornate dalla funzione *fnGetTableDef* (descritta in precedenza al punto 6.1). La mappatura tra i nomi di colonna ritornati dalla funzione *fnGetTableDef* e i segnaposto la si trova all'interno della tabella *FILLER_DEC*, qui di seguito riportata:

Tabella ordinata per FieldName

WildCard	FieldName	AssociatedFunction
$M	castedDenulledFieldName	dbo.fnGetTableDef
$K	castedFieldName	dbo.fnGetTableDef
$^	cSharpPrivateVariableName	dbo.fnGetTableDef
$@	cSharpPublicPropertyName	dbo.fnGetTableDef
$#	cSharpType	dbo.fnGetTableDef
$I	fieldIsIdentity	dbo.fnGetTableDef
$X	fieldIsKey	dbo.fnGetTableDef
$L	fieldLength	dbo.fnGetTableDef
$N	fieldName	dbo.fnGetTableDef
$P	fieldPrecision	dbo.fnGetTableDef
$C	fieldScale	dbo.fnGetTableDef
$T	fieldType	dbo.fnGetTableDef
$F	fullFieldType	dbo.fnGetTableDef
$?	randomData	dbo.fnGetTableDef
$Q	SqlDbType	dbo.fnGetTableDef
$S	stringFieldLength	dbo.fnGetTableDef
$V	variableName	dbo.fnGetTableDef

Tabella ordinata per WildCard

WildCard	FieldName	AssociatedFunction
$#	cSharpType	dbo.fnGetTableDef
$?	randomData	dbo.fnGetTableDef
$@	cSharpPublicPropertyName	dbo.fnGetTableDef
$^	cSharpPrivateVariableName	dbo.fnGetTableDef
$C	fieldScale	dbo.fnGetTableDef
$F	fullFieldType	dbo.fnGetTableDef
$I	fieldIsIdentity	dbo.fnGetTableDef
$K	castedFieldName	dbo.fnGetTableDef
$L	fieldLength	dbo.fnGetTableDef
$M	castedDenulledFieldName	dbo.fnGetTableDef
$N	fieldName	dbo.fnGetTableDef
$P	fieldPrecision	dbo.fnGetTableDef
$Q	SqlDbType	dbo.fnGetTableDef
$S	stringFieldLength	dbo.fnGetTableDef
$T	fieldType	dbo.fnGetTableDef
$V	variableName	dbo.fnGetTableDef
$X	fieldIsKey	dbo.fnGetTableDef

Il livello di sofisticazione cui si può arrivare è limitato solamente dalla fantasia dello sviluppatore che farà uso di questa tecnica (e delle relative funzioni messe a disposizione da DYNACRUD); di seguito si riporta un esempio di templating un po' più articolato:

```
DECLARE @TAB2 char(2) = CHAR(9) + CHAR(9), @TAB3 char(3) = CHAR(9) + CHAR(9) + CHAR(9)
SELECT
(
	REPLACE
	(
		REPLACE
		(
			dbo.fnFillSqlTemplateWithFieldsList
			(
				@TAB2 + '<asp:TemplateField HeaderText="$@" HeaderStyle-HorizontalAlign="Left">' +
CHAR(13) +
				@TAB3 + '<ItemTemplate><asp:Label ID="lbl$@" Text=''<%# Bind("$@") %>''
DataFormatString="{0:F$S}" ApplyFormatInEditMode="true" HtmlEncode="false" runat="server"></asp:Label></ItemTemplate>' +
CHAR(13) +
				@TAB3 + '<EditItemTemplate><asp:TextBox ID="txt$@" MaxLength="$S" Text=''<%#
Bind("$@") %>'' DataFormatString="{0:F$S}" ApplyFormatInEditMode="true" HtmlEncode="false"
runat="server"></asp:TextBox></EditItemTemplate>' + CHAR(13) +
				@TAB3+ '<FooterTemplate><asp:TextBox ID="txt$@"
runat="server"></asp:TextBox></FooterTemplate>' + CHAR(13) +
				@TAB2 + '</asp:TemplateField>' + CHAR(13)

				,'ENTITA_DETT',NULL)
			,'_x'
			,''
		)
		,'_PK'
		,''
	)
)
```

Questo qui sopra è il template utilizzato per creare un GridView, lato ASPX, per un progetto di tipo Web in Visual Studio 2010; di seguito si riporta un estratto del risultato prodotto dall'invocazione di tale template:

```
<asp:TemplateField HeaderText="CodiceEntita" HeaderStyle-HorizontalAlign="Left">
	<ItemTemplate><asp:Label ID="lblCodiceEntita" Text='<%# Bind("CodiceEntita") %>' DataFormatString="{0:F5}" ApplyFormatInEditMode="true" HtmlEncode="false"
runat="server"></asp:Label></ItemTemplate>
	<EditItemTemplate><asp:TextBox ID="txtCodiceEntita" MaxLength="5" Text='<%# Bind("CodiceEntita") %>' DataFormatString="{0:F5}" ApplyFormatInEditMode="true"
HtmlEncode="false" runat="server"></asp:TextBox></EditItemTemplate>
```

```
            <FooterTemplate><asp:TextBox ID="txtCodiceEntita" runat="server"></asp:TextBox></FooterTemplate>
</asp:TemplateField>
<asp:TemplateField HeaderText="CFCreditore" HeaderStyle-HorizontalAlign="Left">
            <ItemTemplate><asp:Label ID="lblCFCreditore" Text='<%# Bind("CFCreditore") %>' DataFormatString="{0:F16}" ApplyFormatInEditMode="true" HtmlEncode="false"
runat="server"></asp:Label></ItemTemplate>
            <EditItemTemplate><asp:TextBox ID="txtCFCreditore" MaxLength="16" Text='<%# Bind("CFCreditore") %>' DataFormatString="{0:F16}" ApplyFormatInEditMode="true"
HtmlEncode="false" runat="server"></asp:TextBox></EditItemTemplate>
            <FooterTemplate><asp:TextBox ID="txtCFCreditore" runat="server"></asp:TextBox></FooterTemplate>
</asp:TemplateField>
…
…
<asp:TemplateField HeaderText="ChiaveARCAPrestazione" HeaderStyle-HorizontalAlign="Left">
            <ItemTemplate><asp:Label ID="lblChiaveARCAPrestazione" Text='<%# Bind("ChiaveARCAPrestazione") %>' DataFormatString="{0:F128}" ApplyFormatInEditMode="true"
HtmlEncode="false" runat="server"></asp:Label></ItemTemplate>
            <EditItemTemplate><asp:TextBox ID="txtChiaveARCAPrestazione" MaxLength="128" Text='<%# Bind("ChiaveARCAPrestazione") %>' DataFormatString="{0:F128}"
ApplyFormatInEditMode="true" HtmlEncode="false" runat="server"></asp:TextBox></EditItemTemplate>
            <FooterTemplate><asp:TextBox ID="txtChiaveARCAPrestazione" runat="server"></asp:TextBox></FooterTemplate>
</asp:TemplateField>
```

Il codice ASPX prodotto dalla presente funzione è perfettamente riconosciuto e correttamente compilato dall'IDE (Integrated Development Environment) di Visual Studio senza che lo sviluppatore debba apportare alcuna modifica al codice risultante (se non quelle, chiaramente, necessarie a creare una versione personalizzata del GridView).
Va sottolineato che il precedente esempio non è esaustivo per la creazione di un GridView completo in C#: per questo compito sono presenti funzioni dedicate nella libreria DYNACRUD (*fnBuildCsharpGridViewASPX*, *fnBuildCsharpGridViewCS* e *fnBuildCsharpGridViewCSS*).

Un'altra caratteristica accattivante della tecnica di "templating" è data dalla possibilità di annidare infiniti statements SQL e, quindi, funzioni come nell'esempio seguente:

```
PRINT (dbo.fnFillSqlTemplateWithFieldsList ('SET @SQL = dbo.fnAddSetParam(@SQL,''$N'', @$N, ''$T'') ' + CHAR(13),'ENTITA_DETT'))

SET @SQL = dbo.fnAddSetParam(@SQL,'[CodiceEntita]', @[CodiceEntita], 'varchar')
SET @SQL = dbo.fnAddSetParam(@SQL,'[CFCreditore]', @[CFCreditore], 'varchar')
SET @SQL = dbo.fnAddSetParam(@SQL,'[CFDebitore]', @[CFDebitore], 'varchar')
…
…
SET @SQL = dbo.fnAddSetParam(@SQL,'[ChiaveARCAPrestazione]', @[ChiaveARCAPrestazione], 'varchar')
```

Come si può vedere, nel "template" (modello) di base è stata creata una dichiarazione per un codice SQL dinamico all'interno del quale è possibile identificare una chiamata ad una funzione (in questo esempio si tratta di una delle funzioni di DYNACRUD, *fnAddSetParam*, ma qualunque altro statement è impiegabile).

Per rendere "funzionante" (a scopo di test) il precedente esempio, questo andrà completato nel seguente modo:

```
PRINT
(
            'DECLARE' + CHAR(13) + '@SQL varchar(MAX) = ''UPDATE ENTITA_DETT''' + CHAR(13) +
            REPLACE
            (
                        dbo.fnFillSqlTemplateWithFieldsList
                        (
                                    ',$V $F = $?' + CHAR(13)
                                    ,'ENTITA_DETT'
                        )
                        ,''''
                        ,''''''
            ) + CHAR(13) +
            dbo.fnFillSqlTemplateWithFieldsList
            (
                        'SET @SQL = dbo.fnAddSetParam(@SQL,''$N'', $V, ''$T'') ' + CHAR(13)
                        ,'ENTITA_DETT',NULL
            ) + CHAR(13) +
            'SET @SQL += CHAR(13) + ''WHERE 1=1''' + CHAR(13) +
            'PRINT(@SQL)'
)
```

La precedente invocazione produrrà il seguente risultato:

```
DECLARE
@SQL varchar(MAX) = 'UPDATE ENTITA_DETT'
,@CodiceEntita varchar(5) = '4NYU4'
,@CFCreditore varchar(16) = 'QXJMANMFBYIAXTB4'
...
...
,@ChiaveARCAPrestazione varchar(128) =
'RU2PNK8KT3SHMWUF6SVK5AUBKF6TTILMSMEJWPSGETZKZPJGCSXXMEUIPJJQIAZJVS5NYIMYVDA7HMNNEATPRJWFRSCBRBYYNZ5I67HPKVPBE8CQEKE4UEVKBRBBE'

SET @SQL = dbo.fnAddSetParam(@SQL,'[CodiceEntita]', @CodiceEntita, 'varchar')
SET @SQL = dbo.fnAddSetParam(@SQL,'[CFCreditore]', @CFCreditore, 'varchar')
...
...
SET @SQL = dbo.fnAddSetParam(@SQL,'[ChiaveARCAPrestazione]', @ChiaveARCAPrestazione, 'varchar')

SET @SQL += CHAR(13) + 'WHERE 1=1'
PRINT(@SQL)
```

Il quale, se copiato, incollato in una nuova scheda di Microsoft SQL Server Management Studio ed eseguito, produrrà a sua volta il risultato seguente:

```
UPDATE ENTITA_DETT
SET [CodiceEntita] = '4NYU4', [CFCreditore] = 'QXJMANMFBYIAXTB4', [CFDebitore] = 'N67NWUIOAIXKXTMR', [CodicePrestazione] = 'WXX3Z', [CodiceSede] = 'RGWYWC',
[CodiceProcedura] = '4TD6', [Progressivo] = 2810, [Anno] = 9302, [Mese] = 'ZD', [AnnoRif] = 1752, [MeseRif] = 'YT', [ImportoCredito] = 744920.27, [ImportoDebito] = 519421.55,
[ImportoSospeso] = 71078.48, [ImportoSospesoInAtto] = 705363.47, [DataInserimento] = 'May 28 2016  1:59PM', [DataUltimaModifica] = 'Apr 27 2016  1:59PM', [CodiceRegione] = 'HZ',
[IdStruttura] = 8499, [ChiaveARCAAnagraficaCodice] = 'VHB', [ChiaveARCAAnagraficaProgressivo] = 4749, [ChiaveARCAPrestazione] =
'RU2PNK8KT3SHMWUF6SVK5AUBKF6TTILMSMEJWPSGETZKZPJGCSXXMEUIPJJQIAZJVS5NYIMYVDA7HMNNEATPRJWFRSCBRBYYNZ5I67HPKVPBE8CQEKE4UEVKBRBBE'
WHERE 1=1
```

Questo è sicuramente l'esempio più evidente del meccanismo di templating ad annidamento progressivo: tramite esso si potrà notare quanto sia semplice e compatto generare complessi costrutti tramite una sola riga di codice SQL; l'ultimo esempio qui sopra riportato è stato suddiviso in più righe per facilitarne la lettura e l'interpretazione ma, scritto tutto di seguito, corrisponde a quanto segue (ovvero tutto in una sola riga):

```
PRINT ('DECLARE' + CHAR(13) + '@SQL varchar(MAX) = ''UPDATE ENTITA_DETT''' + CHAR(13) + REPLACE(dbo.fnFillSqlTemplateWithFieldsList(',$V $F = $?' +
CHAR(13),'ENTITA_DETT'),'''','''''') + CHAR(13) + dbo.fnFillSqlTemplateWithFieldsList('SET @SQL = dbo.fnAddSetParam(@SQL,''$N'', $V, ''$T'') ' + CHAR(13),'ENTITA_DETT') + CHAR(13) +
'SET @SQL += CHAR(13) + ''WHERE 1=1''' + CHAR(13) + 'PRINT(@SQL)')
```

Opportunamente incapsulate, dunque, le chiamate (EXEC) possono essere eseguite dinamicamente.

E' importante sottolineare che la presente funzione fnFillSqlTemplateWithFields altro non è che una chiamata alla funzione esposta di seguito, *fnFillSqlTemplateWithFieldsOrdered*, in cui l'ultimo parametro – quello preposto all'ordinamento dei risultati – è stato impostato a NULL.

E' possibile, in qualunque momento (soprattutto nel caso in cui vengano aggiunti dei segnaposto alla tabella FILLER_DEC) mostrare rapidamente il mapping che verrà eseguito dalla presente funzione fnFillSqlTemplateWithFieldsList tramite il seguente codice di esempio:

```
DECLARE @replacements varchar(max) = ''
SELECT      @replacements = COALESCE(@replacements,'') + FieldName + ' = ' + WildCard + CHAR(13)
FROM        FILLER_DEC
ORDER BY fieldName
PRINT(@replacements)
```

Il quale produce, alla data odierna (quindi, come già sopra riportato, all'attuale popolamento della tabella *FILLER_DEC*), il seguente risultato:

```
castedDenulledFieldName = $M
castedFieldName = $K
cSharpPrivateVariableName = $^
...
...
variableName = $V
```

In modo analogo, per visualizzare rapidamente il mapping di tutte le colonne della tabella di base (nel caso attuale la tabella "ENTITA_DETT) con tutti i segnaposto associati, invocare il seguente codice di esempio e osservarne i risultati:

```
DECLARE @replacements varchar(max) = ''
SELECT      @replacements = COALESCE(@replacements,'') + FieldName + ' = ' + WildCard + CHAR(13)
FROM        FILLER_DEC
ORDER BY fieldName

PRINT
(
            dbo.fnFillSqlTemplateWithFieldsListOrdered
            (
                        @replacements + CHAR(13)
                        ,'ENTITA_DETT'
                        ,NULL
            )
)
```

```
castedDenulledFieldName = ISNULL(@CodiceEntita,'')
castedFieldName = @CodiceEntita
cSharpPrivateVariableName = _codiceEntita_PK
cSharpPublicPropertyName = CodiceEntita_PK
cSharpType = String
fieldIsIdentity = 0
fieldIsKey = 1
fieldLength = 5
fieldName = [CodiceEntita]
fieldPrecision = 0
fieldScale = 0
fieldType = varchar
fullFieldType = varchar(5)
randomData = "ZBIHO"
SqlDbType = SqlDbType.VarChar
stringFieldLength = 5
variableName = @CodiceEntita

castedDenulledFieldName = ISNULL(@CFCreditore,'')
castedFieldName = @CFCreditore
cSharpPrivateVariableName = _cFCreditore_PK
cSharpPublicPropertyName = CFCreditore_PK
cSharpType = String
fieldIsIdentity = 0
fieldIsKey = 1
fieldLength = 16
fieldName = [CFCreditore]
fieldPrecision = 0
fieldScale = 0
fieldType = varchar
fullFieldType = varchar(16)
randomData = "TTPAQNYBVHF8HIQB"
SqlDbType = SqlDbType.VarChar
stringFieldLength = 16
variableName = @CFCreditore
…
…
castedDenulledFieldName = @ChiaveARCAPrestazione
castedFieldName = @ChiaveARCAPrestazione
cSharpPrivateVariableName = _chiaveARCAPrestazione_x
cSharpPublicPropertyName = ChiaveARCAPrestazione_x
cSharpType = String
fieldIsIdentity = 0
fieldIsKey = 0
fieldLength = 128
fieldName = [ChiaveARCAPrestazione]
fieldPrecision = 0
fieldScale = 0
fieldType = varchar
fullFieldType = varchar(128)
randomData =
"GQCMMXH7GPYLKN6JGT4A4PFRSPIKTKCPWU5DWLZWBTSTMIUXYA3GJS4QFYRI6XXH2MOYWF7SPKBCUXYXHUHD45UO7E2DFFTHFHDU5B5XTHTJTS6MWPRRXAZOIVVCKFM"
SqlDbType = SqlDbType.VarChar
stringFieldLength = 128
variableName = @ChiaveARCAPrestazione
```

6.4. Funzione **fnFillSqlTemplateWithFieldsListOrdered** (Scalar-Valued function)

Questa funzione è in realtà la vera funzione di riempimento/rimpiazzo dei segnaposto, nella tecnica di templating: si differenzia dalla precedente *fnFillSqlTemplateWithFieldsList* per il solo fatto di poter ordinare l'elenco dei risultati secondo uno degli elementi segnaposto (fieldName, cSharpType, variableName, etc.); questo può tornare utile, ad esempio, quando si creano elenchi di definizioni delle variabili nel linguaggio C# allo scopo di raggrupparle per tipo di dato.

Un'invocazione di questo tipo ne è un esempio (notare l'ultimo parametro della funzione, impostato per ordinare i risultati sul tipo di dato C# "cSharpType"):

```
PRINT (dbo.fnFillSqlTemplateWithFieldsListOrdered('$# $^;' + CHAR(13),'ENTITA_DETT','cSharpType'))
DateTime _dataInserimento;
DateTime _dataUltimaModifica;
Decimal? _importoCredito_x;
Decimal? _importoDebito_x;
Decimal? _importoSospeso_x;
Decimal? _importoSospesoInAtto_x;
int _idStruttura_PK;
int _progressivo_PK;
int _anno_PK;
int? _annoRif_x;
int? _chiaveARCAAnagraficaProgressivo_x;
String _chiaveARCAPrestazione_x;
String _meseRif_x;
String _mese_PK;
String _codiceEntita_PK;
String _cFCreditore_PK;
String _cFDebitore_x;
String _codicePrestazione_x;
String _codiceSede_PK;
String _codiceProcedura_PK;
String _chiaveARCAAnagraficaCodice_x;
String _codiceRegione;
```

Come si può vedere, l'elenco dei campi è stato ordinato (quindi raggruppato) per il tipo di dato C#: questo potrebbe consentire allo sviluppatore, con pochi passaggi sulla tastiera, di produrre una lista di definizioni di variabili siffatta:

```
DateTime
        _dataInserimento,
        _dataUltimaModifica;
Decimal?
        _importoCredito_x,
        _importoDebito_x,
        _importoSospeso_x,
        _importoSospesoInAtto_x;
int
        _idStruttura_PK,
        _progressivo_PK,
        _anno_PK;
int?
        _annoRif_x,
        _chiaveARCAAnagraficaProgressivo_x;
String
        _chiaveARCAPrestazione_x,
        _meseRif_x,
        _mese_PK,
        _codiceEntita_PK,
        _cFCreditore_PK,
        _cFDebitore_x,
        _codicePrestazione_x,
        _codiceSede_PK,
        _codiceProcedura_PK,
        _chiaveARCAAnagraficaCodice_x,
        _codiceRegione;
```

La qual cosa, innegabilmente, è di una certa utilità per il colpo d'occhio e la manutenzione del codice.

6.5. Funzione **fnAddSetParam** (Scalar-Valued function)

Nel precedente paragrafo, relativo alla funzione *fnFillSqlTemplateWithFieldsList* si è dimostrato come creare sofisticati modelli grazie al "templating" e, negli esempi, è stata utilizzata anche la funzione fnAddSetParam il cui compito è esclusivamente quello di creare dinamicamente un insieme di parametri "SET" da impiegare in sede di UPDATE.

Tali parametri (a differenza del - o in aggiunta al - "semplice" meccanismo di templating) vengono creati solo in virtù della presenza di un valore da assegnare allo statement UPDATE, il che significa che, qualora uno dei valori passati alla funzione non sia stato popolato o contenga un valore NULL, il corrispondente statement "SET" non verrà creato.

Ecco due esempi di invocazione e i relativi risultati:

```
PRINT(dbo.fnAddSetParam('UPDATE ADDIZIONALE_COMUNALE_DETT SET CFCreditore = ''QRCFNC68P15H501D'', ImportoAcconto = 140.21', 'DataInserimento', '2015-10-11 17:35:44', 'DAT'))
UPDATE ADDIZIONALE_COMUNALE_DETT SET CFCreditore = 'RCCFRZ67P13F611D', ImportoAcconto = 140.21, DataInserimento = '2015-10-11 17:35:44'

PRINT(dbo.fnAddSetParam('UPDATE ADDIZIONALE_COMUNALE_DETT SET CFCreditore = ''RCCFRZ67P13F611D'', ImportoAcconto = 140.21', 'DataInserimento', '', 'DAT'))
UPDATE ADDIZIONALE_COMUNALE_DETT SET CFCreditore = 'RCCFRZ67P13F611D', ImportoAcconto = 140.21
```

Notare la differenza tra le due invocazioni: nonostante alla funzione sia stato passato lo stesso numero di parametri, l'ultimo di questi non è stato valorizzato nella seconda invocazione, provocando l'esclusione automatica – dall'insieme di "SET" – del parametro "DataInserimento".

6.6. funzione **fnGetRandomDataByDataType** (Scalar-Valued function)

In tutti quei casi nei quali sia necessario produrre dei valori casuali, ma perfettamente aderenti al tipo di dato della variabile cui essi vanno assegnati, DYNACRUD mette a disposizione nella sua libreria la funzione fnGetRandomDataByDataType.

Benché non sia possibile, avvalendosi della sola tabella di origine "ENTITA_DETT", poter rappresentare tutte le possibilità offerte da questa funzione – a causa della non totalità dei tipi di dato in essa definiti – si fa notare che questa funzione può generare dati casuali – perfettamente validi – per ciascun tipo di dato esistente sia nel linguaggio T-SQL che in C#, ivi compresi i GUID (uniqueidentifiers).

La presente funzione, ogni qualvolta viene invocata, ritorna un nuovo valore casuale, mai generato prima. Da notare, inoltre, che per i dati casuali di tipo stringa (varchar in SQL), non vi sono limiti alle dimensioni impostabili grazie al dimensionamento "MAX", caratteristico dei server SQL a partire dalla versione 2005.

Questi alcuni esempi di invocazione e i relativi risultati (non sarà mai possibile ottenere gli stessi risultati poiché, come precedentemente sottolineato, i dati generati cambiano ad ogni invocazione, quindi quelli qui sotto riportati sono unici):

```
PRINT('BIT = ' + dbo.fnGetRandomDataByDatatype('bit',10,NULL))
PRINT('INT = ' + dbo.fnGetRandomDataByDatatype('int',10,NULL))
PRINT('DECIMAL = ' + dbo.fnGetRandomDataByDatatype('decimal',20,NULL))
PRINT('VARCHAR(100) = ' + dbo.fnGetRandomDataByDatatype('varchar',100,''))
PRINT('DATETIME = ' + dbo.fnGetRandomDataByDatatype('datetime',26,NULL) + ' (Senza delimitatori)')
PRINT('DATETIME = ' + dbo.fnGetRandomDataByDatatype('datetime',26,'') + ' (Con delimitatore SQL)')
PRINT('DATETIME = ' + dbo.fnGetRandomDataByDatatype('datetime',26,'') + ' (Con delimitatore C#)')
PRINT('BINARY = ' + dbo.fnGetRandomDataByDatatype('binary',100,NULL) + ' (Senza delimitatori)')
PRINT('BINARY = ' + dbo.fnGetRandomDataByDatatype('binary',100,'') + ' (Con delimitatore SQL)')
PRINT('BINARY = ' + dbo.fnGetRandomDataByDatatype('binary',100,'') + ' (Con delimitatore C#)')
PRINT('UNIQUEIDENTIFIER = ' + dbo.fnGetRandomDataByDatatype('uniqueidentifier',1,NULL) + ' (Senza delimitatori)')
PRINT('UNIQUEIDENTIFIER = ' + dbo.fnGetRandomDataByDatatype('uniqueidentifier',1,'') + ' (Con delimitatore SQL)')
PRINT('UNIQUEIDENTIFIER = ' + dbo.fnGetRandomDataByDatatype('uniqueidentifier',1,'') + ' (Con delimitatore C#)')
BIT = 1
INT = 180918
DECIMAL = 631210.43
VARCHAR(100) = "IXUVCSGKQHXVI3CDTINHJWS3FKS454RDTI4QXWZHSGGFYK2AUB6VNRIQQMDKJOUUMNS4TYXDUIUSTIHPWVSHPVQDMOMPQ53LCWTF"
DATETIME =  2018-02-18 16:14:27.140  (Senza delimitatori)
DATETIME = '2016-12-22 16:14:27.143' (Con delimitatore SQL)
DATETIME = "2018-03-05 16:14:27.143" (Con delimitatore C#)
BINARY =  0X43666362F533031C  (Senza delimitatori)
BINARY = '0X435C79600DFA3B46' (Con delimitatore SQL)
BINARY = "0X43342216DDCCA284" (Con delimitatore C#)
UNIQUEIDENTIFIER =  190B7A22-E5D7-4465-84B9-9FA24C9B1525  (Senza delimitatori)
UNIQUEIDENTIFIER = '06FC42C4-B28B-4DF1-B8A9-BD12B99FED36' (Con delimitatore SQL)
UNIQUEIDENTIFIER = "316FED0A-0F44-46A1-B4B9-B746A228EED8" (Con delimitatore C#)
```

6.7. funzione **fnCleanVariableName** (Scalar-Valued function)

Non di rado può accadere che alcune tabelle vengano migrate da un ambiente all'altro o, più frequentemente, progettate/disegnate senza tener conto delle best-practices o della miglior naming-convention; conseguentemente ci si può trovare ad assistere a nomi di campo "singolari", contenenti spazi, caratteri speciali, punteggiatura e molto altro.

Ciò, chiaramente, renderebbe pressoché impossibile produrre un mapping dei nomi di campo in nomi di variabili verso altri ambienti/linguaggi (nel nostro caso il C#); per far fronte a questa evenienza, DYNACRUD mette a disposizione la funzione fnCleanVariableName che svolge questa operazione di "pulizia" rimpiazzando, quanto più possibile, quei nomi di colonna che risulterebbero "impropri" in altri contesti, benché "validi" quando racchiusi tra parentesi quadre in ambiente SQL.

I nomi di campo che dovessero contenere più termini separati tra di loro con dei semplici spazi, verranno trasformati in termini separati da simboli di underscore (trattino basso "_") mentre, per gli altri caratteri "singolari", verrà operata una sostituzione tramite il corrispondente codice ASCII, preceduto e seguito sempre da un underscore.

Di seguito alcuni esempi di invocazione e relativi risultati:

```
PRINT (dbo.fnCleanVariableName('@giachiocciolata'))
PRINT (dbo.fnCleanVariableName('Segno_d'archivio'))
PRINT (dbo.fnCleanVariableName('Dati per DC Finanza'))
PRINT (dbo.fnCleanVariableName('Da inserire nella comunica-zione'))
```

```
_64_giachiocciolata
Segno_d_146_archivio
Dati_per_DC_Finanza
Da_inserire_nella_comunica_45_zione
```

6.8. funzione **fnBuildCsharpClass** (Scalar-Valued function)

Tra le caratteristiche più importanti di DYNACRUD vi è sicuramente la capacità di scrivere codice sorgente al posto dello sviluppatore, riducendone drasticamente il tempo di sviluppo e di manutenzione del codice stesso.

Questa peculiarità è portata al massimo livello di espressione nella presente funzione fnBuildCsharpClass (e in quelle successive la cui radice del nome inizia per *fnBuild*).

Attraverso una semplice invocazione di questo tipo:

```
SELECT dbo.fnBuildCsharpClass('ENTITA_DETT') AS EntityClass
```

si assisterà alla creazione di circa 1000 (mille) righe di codice C# nativo (compatibile con il framework .NET 3.5 e successive versioni) componenti una classe (nell'accezione della programmazione ad oggetti) completa di tutte le variabili (opportunamente mappate/ricodificate secondo quanto documentato nelle precedenti funzioni) nonché i metodi, sia pubblici che privati, necessari a gestire una CRUD nella sua totalità (eccezion fatta per lo strato "presentation", per il quale – in minima parte – sono di ausilio le funzioni *fnBuildCsharpGridViewASPX*, *fnBuildCsharpGridViewCS*, *fnBuildCsharpGridViewCSS*, tutte preposte alla generazione di codice sorgente C# per la gestione delle GridViews).

Una volta che l'invocazione è avvenuta, basterà copiare il codice dal riquadro dei risultati di Microsoft SQL Server Management Studio ed incollarlo in una scheda di Visual Studio (preposta al recepimento di una classe C#) ed eseguire la compilazione che risulterà perfettamente idonea all'ambiente, in qualunque tipo di architettura a 32 o 64 bit.

Ciò che viene inserito in questa classe, ad eccezione della definizione delle variabili private, delle proprietà pubbliche, dei costruttori e del metodo pubblico "SelInsUpdDel" (tutti elementi creati dinamicamente in tempo reale dall'impiego combinato delle funzioni della libreria di DYNACRUD), proviene dalla tabella *SourcesRepository,* all'interno della quale sono state inserite le funzionalità corrispondenti a vari metodi – pubblici o privati – che sono stati ritenuti di varia utilità (ivi compreso il codice di un blocco "Main" di test per un'applicazione di tipo Console, da copiare e incollare dentro la classe Program.cs).

Queste funzionalità potranno essere modificate, eliminate o aggiunte a piacimento, senza soluzione di continuità, e verranno - sempre e comunque - riflesse all'interno della classe qualora presenti nella tabella *SourcesRepository*.

Ovviamente si consiglia, prima di apportare modifiche o aggiunte a questa tabella, di testare ogni singola funzione/metodo in un ambiente dedicato allo scopo di validarne la sintassi e il comportamento logico: allo stato dell'arte – e dunque alla data di redazione del presente documento – le funzioni/metodi presenti nella tabella *SourcesRepository* sono risultate perfettamente funzionanti in tutti i casi di test.

Come è naturale dedurre, con pochi, semplici accorgimenti e manipolazioni, questa funzione può essere adattata/replicata affinché produca qualunque oggetto (classe) in qualunque linguaggio di programmazione, apportando le dovute modifiche/aggiunte.

6.9. funzione **fnBuildStoredProcedureForCRUD** (Scalar-Valued function)

Anche questa funzione, come le altre il cui prefisso è *fnBuild*, si occupa di scrivere codice sorgente al posto dello sviluppatore; in questo caso però – unico in tutto il sistema DYNACRUD – una funzione scritta in codice T-SQL scrive una stored procedure anch'essa in T-SQL: si tratta della stored procedure di base di tutto il sistema DYNACRUD. Attraverso questa stored procedure, il cui prefisso è spInsUpdDel vengono effettuate le principali operazioni di Inserimento, Modifica, Cancellazione e Lettura dei dati sulla/dalla tabella (o vista) in corso di gestione.

Ogni qualvolta viene creata una tabella che dovrà essere gestita dal sistema DYNACRUD, sarà obbligatoriamente necessario invocare la presente funzione allo scopo – appunto – di generare la stored procedure attorno alla quale girano tutte le funzionalità pertinenti una CRUD.

Se, nel tempo, alcune colonne/campi dovessero essere aggiunti/modificati/rimossi dalla tabella, sarà necessario invocare nuovamente questa funzione (nonché tutte le altre funzioni il cui prefisso è *fnBuild*) ed incollare i risultati di queste negli appositi ambiti (editor SQL, editor di codice C#, editor di codice ASPX, a seconda della funzione invocata); questo consentirà di mantenere allineato tutto il codice dell'applicazione in tempi ridottissimi e con un intervento minimo richiesto allo sviluppatore.

ATTENZIONE!!! Le operazioni di copia e incolla sono distruttive: tutto ciò che andremo a rimpiazzare con il codice appena generato, sovrascriverà in modo irreversibile quanto presente allo stato pre-incollaggio (a meno di un ritorno alle condizioni originarie tramite UNDO e/o utilizzando sistemi di repository del codice sorgente).

Questo avvertimento è valido per quei casi in cui, una volta prodotto il codice sorgente di "base", vengono apportate considerevoli variazioni ai moduli generati automaticamente (ad esempio le classi-base, i GridViews in aspx, gli eventi per i GridViews e quant'altro generato dal presente sistema DYNACRUD).

Se lo sviluppatore saprà mantenere ben distinti e separati i moduli di gestione - dallo strato presentation a quello business – lasciando pressoché inalterato il codice generato automaticamente da DYNACRUD, sarà praticamente possibile sostituire, in una sola manciata di minuti, tutto l'impianto di base senza alcuno sforzo.

Questa funzione si invoca nel seguente modo:

```
SELECT dbo.fnBuildStoredProcedureForCRUD('ENTITA_DETT') AS StoredProcDef
```

e produce la stored procedure COMPLETA per la gestione di una CRUD.

Altre funzioni ancillari, di minor importanza (per lo sviluppatore), sono presenti nel sistema DYNACRUD e da queste l'intero sistema non può prescindere in quanto, in una porzione o nell'altra del sistema stesso, sono collegate da un legame di dipendenza con le altre funzioni per cui, benché potrebbe non essere mai necessario invocarle direttamente, queste svolgono un ruolo fondamentale per il corretto funzionamento dell'intero impianto: NON RIMUOVERE arbitrariamente nessuna di queste funzioni, anche se non citate e/o documentate in questa presentazione, poiché il sistema DYNACRUD ne risulterebbe seriamente compromesso, smettendo di funzionare.