IL PRIMO MAGAZINE ITALIANO DEDICATO A VISUAL BASIC, C#, .NET, ACCESS E SQL SERVER

Poste Italiane S.p.A. - Spedizione in Abbonamento Postale - D.L. 353/2003 conv. in L. 27/02/2004 n°46 art.1,comma 1, DCB PERUGIA

VISUAL BASIC & .NET JOURNAL

n.63

Maggio/Giugno 2005 - bimestrale - Anno 11 - Euro 7,75

ASP.NET

Infrastruttura di sicurezza
Validazione dell'input

Yukon in scena:
si alza il sipario

PRODOTTI

C-Sharpener
Sharp Shooter
VBDocman

Database

Log delle transazioni mediante trigger
Access, MDAC e Windows 98: problemi di compatibilità

Architect's Corner

Ottimizzare i lock sui database

Software Engineering

I design pattern più famosi in VB
Logica applicativa in VB .NET

GRUPPO EDITORIALE INFOMEDIA

Alla scoperta di Visual Basic e VB .NET

L'obiettivo del libro è di permettere a chiunque di iniziare a sviluppare i primi programmi in Visual Basic. La trattazione prevede un linguaggio molto semplice ed è supportata da esempi ed esercizi, che hanno lo scopo di dare un riscontro pratico agli argomenti esposti. Il lettore è guidato in modo graduale durante l'apprendimento; inizia dapprima ad acquisire confidenza con i concetti di base della programmazione e prosegue imparando a sviluppare delle applicazioni sempre più complesse, aumentando la propria conoscenza fino a raggiungere la capacità di gestire, fra l'altro, dei contenuti multimediali e degli archivi organizzati in basi di dati.
Gli ultimi capitoli sono incentrati sulla nuova versione del linguaggio, Visual Basic .NET, puntando l'attenzione in particolar modo sulle diversità che caratterizzano questa versione rispetto alla precedente. Sono mostrati, attraverso una serie di esempi pratici, i punti chiave del nuovo linguaggio oltre a come implementare una semplice soluzione basata su Visual Basic .NET

Alla scoperta di Visual Basic e VB .NET
Maurizio Crespi - Andrea Frosini
pagg. 385
ISBN 8881500108
€ 34.00

dal SOMMARIO

Controllo If e varianti
Controllo Select Case
La struttura For
Strutture regolate da una condizione booleana
La gestione dei file di testo
I file ad accesso casuale
Applicazioni in grado di far uso dei databse
Oggetto Recordset: ricerche all'interno degli archivi Interrogare i database
Produzione di stampe di elevata qualità
La visualizzazione di immagini grafiche
Disegnare immagini con Picture
La creazione dei menu
Gestire i menu a run time
La gestione degli errori
Sviluppare codice migliore
Applicazioni SDI e MDI
I file help di Windows
La programmazione ad oggetti
VB .NET
VB .NET e le WebForm
ADO .NET e "Data Components"
DataSet, DataTable e ReportViewer

GRUPPO EDITORIALE INFOMEDIA
THE SOFTWARE SIDE OF YOUR MIND

Clicca su www.infomedia.it per leggere il capitolo d'esempio

EDITORIALE

La supremazia dei blog

Alla fine non ce l'ho fatta. Alla fine, anche io ho inaugurato il mio blog. Per essere più precisi, insieme ai collaboratori fissi di *www.dotnet2themax.it* abbiamo deciso di aprire un "team blog", dove ognuno di noi può postare tip, piccoli pezzi di codice, raccontare le proprie vicissitudini quotidiane o anche solo commentare quello che accade nel mondo della programmazione. Devo dire che mi sono subito appassionato a questo strumento, e di solito riesco a postare almeno 3–4 volte a settimana, con la consolazione che quando non sono io a farlo c'è qualcun altro del team che ha qualcosa di interessante da dire.

Non siamo stati certo i primi autori ad aprire un blog, nè in Italia nè tantomeno nel resto del mondo. Oramai sembra che tutte le informazioni tecniche più interessanti siano disponibili su qualche blog, in particolare su quelli di qualche team di sviluppo in Microsoft Corp. Spesso la qualità dei post sul blog supera quella degli articoli presenti su riviste (cartacee o online) ed è l'unico modo per mantenersi aggiornati (esclusa naturalmente la lettura di Visual Basic & .NET Journal… :–). Se volete sapere cosa accade nel mondo che ruota intorno a .NET, una visita a *http://scoble.weblogs.com/* spesso è più fruttuosa della lettura di qualche seriosa analisi.

Tutto bene, quindi? Non proprio. Perchè – anche a costo di andare controcorrente e di smentire quanto ho appena detto – non sono affatto sicuro che i blog siano lo strumento migliore per condividere la conoscenza nel settore IT in generale e della programmazione in particolare. Il fatto è che tutta questa enorme massa di informazioni è buttata lì sul Web a disposizione di tutti, ma se non fosse per Google non riusciremmo mai a trovarla. E a volte Google non basta neanche, e mi è capitato spesso di trovare solo per puro caso alcune "chicche" che avrebbero meritato di apparire al primo posto nei risultati dei motori di ricerca.

Queste informazioni dovrebbero essere non solo indicizzate, ma anche organizzate in qualche modo, per poterle recuperare velocemente. Un tentativo in questa direzione arriva da Microsoft con l'iniziativa CodeZone (*www.codezone.com*), che quando arriverà a maturità *dovrebbe* (il condizionale è d'obbligo) fungere da aggregatore di articoli tecnici e altre risorse disponibili sul Web. Ma il limite di CodeZone, a mio parere, è che l'iniziativa è lasciata ai singoli autori (che devono ricordarsi di inviare a CodeZone un link ai propri articoli).

Non so se esiste una soluzione. Forse Microsoft dovrebbe investire seriamente in una specie di index (stile Yahoo o Virgilio, per intenderci) combinato con un motore tipo MSN Search per permettere di rintracciare velocemente quello che ci serve. Fino ad allora, continuerò a scrivere più spesso che posso sul mio piccolo blog e a consultare periodicamente i blog e i siti che più mi interessano.

Francesco Balena
fbalena@codearchitects.com

VISUALBASIC & .NET JOURNAL
online.infomedia.it
n. 63 - maggio/giugno 2005
bimestrale - anno undicesimo

Direttore Responsabile
Marialetizia Mari *(mmari@infomedia.it)*

Direttore Esecutivo
Francesco Balena *(fbalena@infomedia.it)*

Managing Editor
Renzo Boni *(rboni@infomedia.it)*

Collaboratori
Marco Bellinaso
Andrea Benedetti
Gionata Aladino Canova
Marco Caridi, Fabio Perrone
Paolo Pialorsi
Francesco Quaratino
Ingo Rammer, David Stanley
Lorenzo Vandoni

GRUPPO EDITORIALE INFOMEDIA

Direzione
Natale Fino *(nfino@infomedia.it)*

Marketing & Advertising
Segreteria: 0587/736460
marketing@infomedia.it

Amministrazione
Sara Mattei
(amministrazione@infomedia.it)

Grafica
Manola Greco *(mgreco@infomedia.it)*

Technical Book
Lisa Vanni (*book@infomedia.it)*

Segreteria
Enrica Nassi
(info@infomedia.it)

Stampa
TIPOLITOGRAFIA PETRUZZI
Citta' di Castello (PG)

Ufficio Abbonamenti
Tel. 0587/736460 – Fax 0587/732232
e–mail: abbonamenti@infomedia.it
www.infomedia.it

Gruppo Editoriale Infomedia srl
Via Valdera P., 116 – 56038 Ponsacco (PI) Italia
Tel. 0587/736460 – Fax 0587/732232
red_vbj@infomedia.it
Sito Web www.infomedia.it

Visual Basic Journal *è una rivista di Gruppo Editoriale Infomedia S.r.l. Via Valdera P, 116 Ponsacco – Pisa.*
Registrazione presso il Tribunale di Pisa n. 20/1999


Your potential. Our passion.
Microsoft
Sentiti libero di pensare in grande.
Con Visual Studio .NET 2003 aumenti la tua produttività scrivendo meno codice che in passato, così puoi trasformare le tue idee in realtà più velocemente di quanto hai mai immaginato. Hai un ambiente di sviluppo efficiente e flessibile, il supporto per Windows e Web Form, un nuovo sistema di installazione che ti aiuta ad eliminare il problema della registrazione e della condivisione di DLL:
microsoft.com/italy/vstudio/value/
Per scoprire tutte le potenzialità dello strumento che ti aiuterà a realizzare le tue idee e i tuoi progetti partecipa ai webcast su tecnologia Live Meeting:
microsoft.com/italy/webcast_msdn/
Microsoft
Visual Studio

SOMMARIO

MAGGIO/GIUGNO

N.63

SPECIALE

SOFTWARE ENGINEERING

DATABASE

RUBRICHE

PRODOTTI

ARCHITECT'S CORNER

ENTERPRISE

Codice allegato

All'indirizzo ftp.infomedia.it/pub/VBJ *sono liberamente scaricabili tutti i listati relativi agli articoli pubblicati. La presenza di questa immagine indica l'ulteriore disponibilità, allo stesso indirizzo, di un progetto software relativo all'articolo in cui l'immagine è inserita. Il nome identifica la cartella sul sito ftp.*

ASP.NET: alla base della sua sicurezza

Valutiamo alcuni aspetti portanti della infrastruttura di sicurezza di ASP.NET.

di Paolo Pialorsi

Il motore di ASP.NET e il .NET Framework, sul quale esso si basa, forniscono diversi strumenti per garantire un'infrastruttura sicura di esecuzione del codice. Molto spesso quando si pensa alla sicurezza in ASP.NET vengono in mente la Forms Authentication, la Windows Authentication e Passport. Se poi si pensa ad ASP.NET 2.0 ricorrono anche concetti come "Profiling", "Personalization and Membership" e "Role Management". In questo articolo parleremo degli aspetti relativi alla sicurezza di ASP.NET che sono alla base di tutti questi servizi.

Security Principal e Identity

Prima di addentrarci nei veri e propri contenuti dell'articolo è opportuno chiarire alcuni concetti fondamentali e imprescindibili. Ogni processo eseguito dal sistema operativo Windows (penso a NT/2000/XP/2003) è associato ad un Security Context che è rappresentato da un token. Il token identifica credenziali a livello di sistema.

Queste credenziali vengono generalmente dette Security Principal e possono rappresentare: un utente, una macchina o un servizio.

Inoltre, pensando per semplicità al solo concetto di utente, sappiamo che ogni utente appartiene a qualche gruppo. Nel .NET Framework il concetto di Security Context è rappresentato in forma astratta da due interfacce: IIdentity e IPrincipal.

Paolo Pialorsi *è un consulente e autore specializzato nello sviluppo di Web Service e soluzioni Web con il Framework .NET di Microsoft. Lavora nell'omonima società Pialorsi Sistemi S.r.l. e fa parte del gruppo DevLeap. Può essere contattato via email: paolo@devleap.it. Paolo mantiene un blog all'indirizzo: http://blogs.devleap.com/paolo/ .*

Il singolo utente viene associato dal sistema a un'identità e in .NET corrisponde a una classe che implementa l'interfaccia IIdentity.

Una identità, unita ai gruppi/ruoli ai quali appartiene, viene detta Principal e in .NET è rappresentata da una classe che implementa l'interfaccia IPrincipal.

C#

```
public interface IIdentity
{
bool IsAuthenticated { get; }
string AuthenticationType { get; }
string Name { get; }
}

public interface IPrincipal
{
bool IsInRole(string role);
IIdentity Identity { get; }
}
```

Nel .NET Framework 1.1 abbiamo già a disposizione alcune implementazioni di queste interfacce:

- WindowsIdentity e WindowsPrincipal: rappresentano l'identità di un utente Windows, di macchina o di dominio, e l'associazione tra l'identità Windows e i gruppi di Windows ai quali essa appartiene.

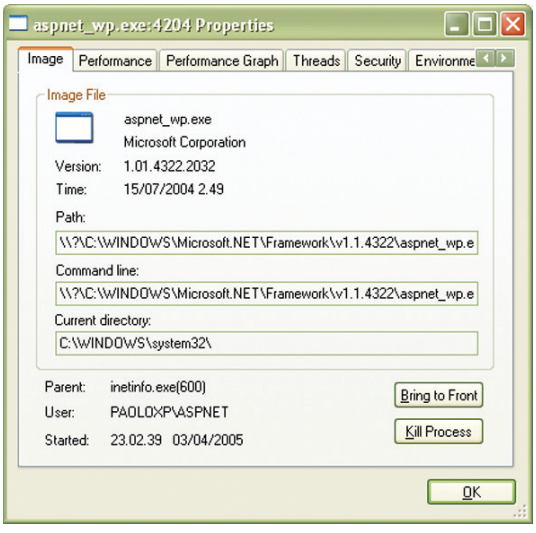


Figura 1 Il processo di ASP.NET in Windows XP/2000

- GenericIdentity e GenericPrincipal: rappresentano una identità personalizzata, slegata dal sistema e dall'eventuale dominio Windows, e l'associazione tra tale identità ed eventuali ruoli applicativi personalizzati, definiti come array di stringhe.
- FormsIdentity: costituisce l'identità di un utente autenticato in ASP.NET utilizzando la Forms Authentication. Questa identità può essere associata a ruoli personalizzati per dare origine a un GenericPrincipal.
- PassportIdentity: costituisce l'identità di un utente autenticato in ASP.NET utilizzando Microsoft Passport.
  Anche questa identità può essere associata a ruoli personalizzati per sfociare in un GenericPrincipal.

Durante l'esecuzione di un processo .NET possiamo chiedere al Framework, invocando il metodo WindowsIdentity.GetCurrent, di comunicarci qual è il Security Context di esecuzione del thread corrente, ottenendo come risposta un oggetto di tipo WindowsIdentity.

Parlo di thread corrente in quanto ogni processo, come ho già detto, viene lanciato con un suo Security Context, ma all'interno del processo sono i thread ad eseguire il codice.

Nella più semplice delle situazioni avremo un processo mono-thread, nel quale il singolo thread girerà usando le credenziali associate al token del processo che lo ospita. Nel caso di processi multi-thread di norma tutti i thread sono eseguiti utilizzando il Security Context del processo ospite.

Nessuno ci vieta però di creare dei thread che impersonifichino delle identità differenti.

Questo spesso si verifica proprio nelle applicazioni che implementano dei servizi server. In questo modo infatti noi possiamo avere un processo server che gira con determinate credenziali e che poi impersonifica di volta in volta i vari client che vi si connettono, utilizzando dei thread dedicati alle singole richieste.

Anche il motore di ASP.NET lavora in questo modo, ecco perchè ne abbiamo parlato.

Identità del processo ASP.NET

Il motore di ASP.NET è di norma ospitato da Internet Information Services di Windows. A seconda della versione di Windows possono però cambiare alcune configurazioni relative all'identità del processo.

> Ogni processo eseguito dal sistema operativo Windows è associato ad un Security Context che è rappresentato da un token

Partiamo da Windows 2000 Server e Windows XP, cioè occupiamoci di IIS 5.0/5.1. Nel caso di IIS 5.x il processo di ASP.NET è un unico processo, che si chiama ASPNET_WP.EXE (**Figura 1**) e che gira, per impostazione predefinita, utilizzando un Security Context associato all'identità [Nome Macchina]\ASPNET.

Questa impostazione è dichiarata nel file di configurazione di .NET Framework (machine.config), nel tag processModel:

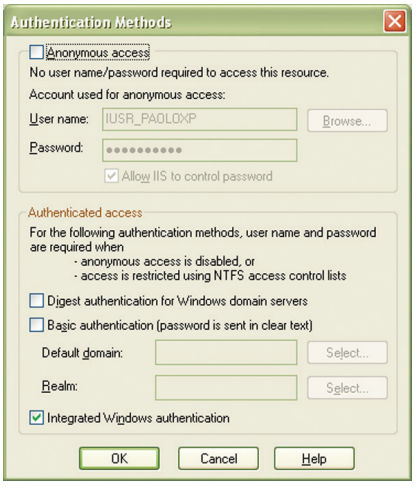


Figura 2 Pannello di configurazione dell'autenticazione di IIS

MACHINE.CONFIG

```
<processModel ... userName="machine"
password="AutoGenerate" ...
```

Con queste impostazioni predefinite la password dell'utente ASPNET viene autogenerata, criptata e salvata nel registro di sistema all'interno della chiave HKLM\SECURITY\Policy\Secrets\aspnet_WP_PASSWORD, cioè nel cosiddetto LSA (Local Security Authority) Secret Store di Windows. In Windows Server 2003, grazie alle novità introdotte nel motore di IIS 6.0, possiamo avere più processi, di nome W3WP.EXE, che girano in parallelo e che rappresentano differenti Application Pool.

Ciascun Application Pool può essere configurato per utilizzare una identità specifica; inoltre ogni Application Pool può sfruttare più processi paralleli, consentendoci di avere un semplice e rapido meccanismo di bilanciamento di carico (load balancing) tra processi su di una stessa macchina server. Di norma il processo W3WP.EXE gira con l'identità "NT AUTHORITY\NETWORK_SERVICE".

Identità del client

A questo punto si presentano N possibili configurazioni relative all'identità del client che rivolge richieste al motore di ASP.NET. Esaminiamo ad esempio la configurazione predefinita di ASP.NET e IIS. La configurazione predefinita di IIS prevede accesso anonimo e autenticazione integrata attivi. ASP.NET nel machine.config (configurazione di .NET a livello di macchina) e nel web.config (configurazione di ASP.NET a livello di applicazione o directory virtuale) di norma prevede l'autenticazione Windows, senza alcun tipo di impersonificazione del client.

A questo punto se un browser richiede una pagina al motore di ASP.NET, la richiesta passerà prima di tutto attraverso IIS, il quale collegherà la richiesta all'account generico IUSR_[Nome Macchina]. Tale account sarà fornito al motore di ASP.NET come identità (WindowsIdentity) della richiesta, marcata come non autenticata (proprietà IsAuthenticated della WindowsIdentity pari a false). Il motore di ASP.NET eseguirà la richiesta impersonificando l'utente associato al Security Context del processo ASP.NET e non l'identità del client, in quanto l'impersonificazione del client è disattivata nelle impostazioni predefinite.

Ciò significa che sarà l'utente [Nome Macchina]\ASPNET o "NT AUTHORITY\NETWORK_SERVICE" ad essere utilizzato per accedere qualunque tipo di risorsa di sistema. Eseguendo il seguente codice all'interno di una pagina:

C#

```
private void Page_Load(object sender, System.EventArgs e)
{
WindowsIdentity currentIdentity = WindowsIdentity.GetCurrent();
WindowsIdentity userIdentity = (WindowsIdentity)User.Identity;

System.Text.StringBuilder sb = new System.Text.StringBuilder();
sb.Append("<br><b>Current Process Identity</b>");
sb.Append("<br>Username: ");
sb.Append(currentIdentity.Name);
sb.Append("<br>IsAuthenticated: ");
sb.Append(currentIdentity.IsAuthenticated);
sb.Append("<br>IsAnonymous: ");
sb.Append(currentIdentity.IsAnonymous);
sb.Append("<br> <br><b>Current User Identity</b>");
sb.Append("<br>Username: ");
sb.Append(userIdentity.Name);
sb.Append("<br>IsAuthenticated: ");
sb.Append(userIdentity.IsAuthenticated);
sb.Append("<br>IsAnonymous: ");
sb.Append(userIdentity.IsAnonymous);
```

```
securityInfos.Text = sb.ToString();
}
```

Se pensiamo che la variabile securityInfos sia una Label, nel browser avremo il seguente testo:

Current Process Identity
Username: [Nome Macchina]\ASPNET
IsAuthenticated: True
IsAnonymous: False
Current User Identity
Username:
IsAuthenticated: False
IsAnonymous: True

Il codice è stato eseguito su un sistema Windows XP. Come si vede l'utente associato al processo di ASP.NET è [Nome Macchina]\ ASPNET, mentre l'utente associato alla richiesta non ha nome, infatti IUSR_[Nome Macchina] non ha nome, e risulta non autenticato, quindi anonimo. In questo caso le risorse di sistema accedute da ASP.NET, per esempio file, database e gli stessi file ASPX, saranno verificate rispetto ai diritti dell'utente ASPNET.

Proviamo ora ad attivare l'autenticazione Windows in IIS, cioè togliamo la spunta dal flag di accesso anonimo alla nostra applicazione (**Figura 2**). Il risultato nel browser sarà:

Current Process Identity
Username: [Nome Macchina]\ASPNET
IsAuthenticated: True
IsAnonymous: False

Current User Identity
Username: [Nome Macchina]\PaoloPi
IsAuthenticated: True
IsAnonymous: False

Ecco che l'utente con il quale sto eseguendo la richiesta dal browser, sfruttando l'autenticazione integrata di Windows, viene passato al motore di ASP.NET, il quale esegue ancora la richiesta nel Security Context dell'utente associato al processo di ASP.NET, ma sarà ora in grado di sapere che l'utente connesso è PaoloPi.

Eventuali accessi a risorse, database e file, compresi gli ASPX, saranno comunque sempre eseguiti dall'utente del processo di ASP.NET, che in questo caso per noi è [Nome Macchina]\ ASPNET.

Impersonificare un'identità

Attiviamo ora l'impersonificazione dell'utente in ASP.NET.

Possiamo farlo agendo sul file web.config della nostra applicazione di prova, aggiungendo o modificando il tag identity:

WEB.CONFIG

```
<identity impersonate="true" />
```

Rieseguendo adesso la pagina, avremo ancora un diverso risultato:

Current Process Identity
Username: [Nome Macchina]\PaoloPi
IsAuthenticated: True
IsAnonymous: False

Current User Identity
Username: [Nome Macchina]\PaoloPi
IsAuthenticated: True
IsAnonymous: False

A questo punto il thread che esegue questa singola richiesta, pur girando all'interno del processo di ASP.NET, utilizzerà un Security Context specifico e relativo all'identità dell'utente PaoloPi, cioè quella dell'utente che sta eseguendo la richiesta dal browser.

Nel caso in cui l'utente non abbia accesso ad eventuali risorse accedute (ricordo ancora una volta che in queste sono comprese anche le pagine ASPX stesse) avremo un errore di tipo "Access Denied" (**Figura 3**).

> L'impersonificazione in Windows 2000 è consentita solo se l'account del processo ASP.NET ha il privilegio "Act as Operating System"

L'impersonificazione in Windows 2000 è consentita solo se l'account del processo ASP.NET ha il privilegio "Act as Operating System", impostabile dalle Local Security Policy della macchina (secpol.msc).

In Windows Server 2003 e in Windows XP questa esigenza è stata superata. Possiamo anche decidere di impersonificare sempre e comunque un particolare utente applicativo, indicando nel web.config una specifica identità da impersonificare:

WEB.CONFIG

```
<identity impersonate="true" userName="[Nome Macchina]\Utente_
                    Applicativo" password="pwd_utente" />
```

Con questa nuova configurazione avremo:

Current Process Identity
Username: [Nome Macchina]\Utente_Applicativo
IsAuthenticated: True
IsAnonymous: False

Current User Identity
Username: [Nome Macchina]\PaoloPi
IsAuthenticated: True
IsAnonymous: False

Dovrebbe essere abbastanza chiaro il fatto che a questo punto il processo di ASP.NET girerà con il Security Context di ASPNET, la richiesta sarà identificata come proveniente dall'utente PaoloPi e il thread che esegue la richiesta sarà associato all'identità custom ([Nome Macchina]\Utente_Applicativo) che abbiamo deciso di impersonificare nel web.config.

A volte è necessario eseguire la maggior parte delle richieste con il Security Context standard, tranne quelle che vogliamo eseguire con un contesto differente, di solito con privilegi maggiori e proprio per questo da usare solo temporaneamente.

Una possibilità che abbiamo, ma della quale non vedremo tutto il codice in questo articolo, è quella di impersonificare un utente specifico, utilizzando API di Windows per creare un token relativo all'utente da impersonificare, creando poi un'istanza della classe WindowsImpersonationContext, tramite il metodo Impersonate della classe WindowsIdentity.

C#

```
// Creo un token utilizzando l'API
// LogonUser di advapi32.dll
// ...

// Impersonifico il token
WindowsImpersonationContext impersonationContext = Windows
                                Identity.Impersonate(tokenPtr);
// Qui eseguo codice che gira con
// l'identità dell'impersonationContext
// ...

// Esco dal contesto di impersonificazione
// e torno al Security Context standard
impersonationContext.Undo();
```

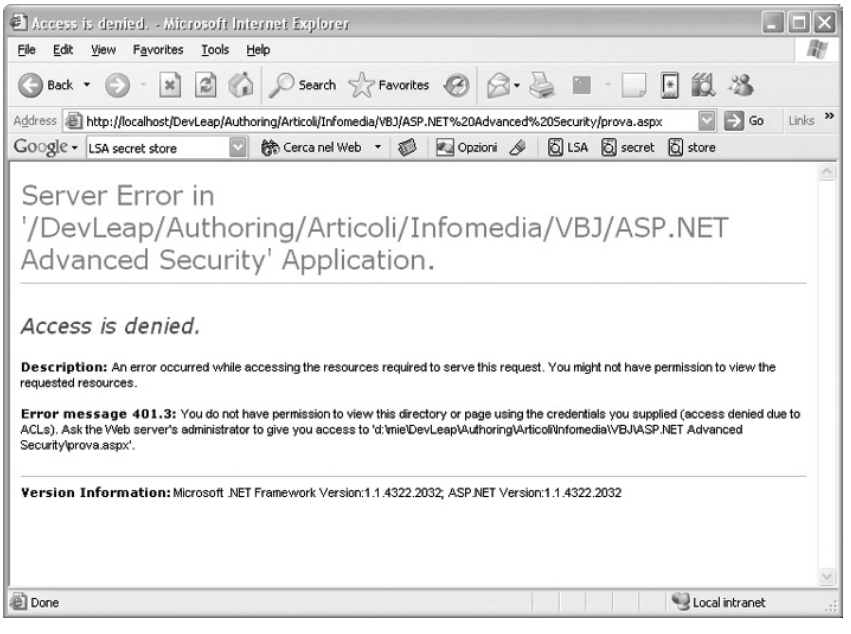


Figura 3 Errore "Access Denied" in caso di utente non autorizzato

Per ulteriori dettagli su tutte le possibili configurazioni di ASP.NET e IIS rimando alla lettura del capitolo 2 del libro "ASP.NET Full Contact" [2].

Livelli di Trust

Un altro aspetto fondamentale e portante dell'infrastruttura di sicurezza di ASP.NET, a partire dalla versione 1.1, sono i livelli di trust per l'esecuzione del codice.

Il .NET Framework basa la sua sicurezza sulla Code Access Security. Code Access Security prevede un meccanismo di identificazione degli assembly che devono essere caricati in memoria ed eseguiti, al fine di stabilire se questi hanno o meno il permesso di svolgere determinate attività sul PC nel quale vengono eseguiti, a prescindere da quelli che sono i permessi dell'utente correntemente autenticato sulla macchina. Per esempio io posso utilizzare il mio PC con un utente che ha diritto di accesso al disco C:\ della mia macchina, ma posso impedire a tutti gli assembly che provengono dalla rete Internet o Intranet di accedere al disco C:\, anche se sono io ad eseguirli.

Questo consente di raggiungere dei livelli di sicurezza maggiori rispetto alla classica esecuzione di un programma nello stesso contesto di sicurezza di chi lo esegue, come accade per esempio con un programma Visual Basic 6, con uno script VBS/JS o anche con un programma C++ unmanaged. Generalmente i criteri di identificazione degli assembly .NET si basano sulla loro provenienza fisica (local machine, Intranet, Internet, URL specifiche), sul produttore (eventualmente tramite Authenticode), sullo strong name [4], ecc.

Una volta identificata la provenienza e le caratteristiche di un assembly (si parla di "evidence", cioè "prove") si costruisce un elenco di permessi associati all'assembly da caricare in memoria ed eseguire, applicando un meccanismo di policy a più livelli, il cui risultato sono i permessi complessivi per quell'assembly [5]. Un livello di trust non è altro che un insieme di permessi ai quali deve sottostare un'applicazione ASP.NET durante la sua esecuzione. Riferendosi a questi concetti si parla spesso di "sandboxing". La configurazione predefinita di ASP.NET 1.1 prevede i seguenti livelli di trust:

- Full: è il livello predefinito (purtroppo!) e prevede qualsiasi tipo di permesso in quanto, di fatto, spegne la verifica dei permessi.
- High: livello di permessi molto alto, consente comunque di fare quasi tutto sulla macchina. Rispetto al livello Full Trust per esempio non consente l'esecuzione di codice unmanaged (per es. chiamate a componenti COM).
- Medium: maggiormente restrittivo rispetto al precedente. Ad esempio non consente l'accesso al file system, tranne che per i folder dell'applicazione corrente, nei quali è consentito l'accesso in lettura, scrittura e aggiunta. Consente l'uso dell'IsolatedStorage [5].
- Low: restrizione del livello Medium che consente l'accesso in sola lettura ai folder dell'applicazione, non consente l'accesso a database esterni. Consente l'uso dell'IsolatedStorage [5] con una quota di 1 Mbyte.
- Minimal: consente solo l'esecuzione di codice ASP.NET, ma impedisce l'accesso a database, file system, isolated storage, socket, web service, ecc.

Ciascuno di essi corrisponde a dei file .config presenti nella cartella %windir%\Microsoft.NET\Framework\v1.1.4322\CONFIG.

Nel file machine.config, piuttosto che nel web.config di una particolare applicazione web, possiamo configurare un livello di trust nel modo seguente:

C#

```
<system.web>
<trust level="Minimal" originUrl="" />
</system.web>
```

Il parametro originUrl indica una eventuale URL da considerare fruibile nel caso di permessi di accesso a risorse via rete, come permessi relativi ai Socket e alle WebRequest.

Una buona regola da seguire, in particolare quando si svolge attività di hosting di applicazioni ASP.NET di terzi, sarebbe abbassare al minimo indispensabile il livello di trust delle applicazioni.

Questo per evitare che eventuali "code injection" possano svolgere attività da noi non contemplate, ma potenzialmente molto rischiose per il nostro server e/o per la sicurezza delle altre applicazioni ospitate.

Pensate che cosa potrebbe accadere, se dovessimo pubblicare su di un server che svolge attività di web hosting di massa, una pagina ASPX che si mette a sfogliare il contenuto del

file system del server, magari consentendoci di cancellare dei file o mostrando il contenuto dei web.config di tutte le altre applicazioni web in hosting su quel server. Potenzialmente, in assenza di controlli tramite le ACL (Access Control List) di Windows, potremmo creare dei seri problemi a quel server e alle sue applicazioni.

D'altra parte impostare le ACL su ogni folder applicativo sarebbe molto oneroso e richiederebbe l'intervento di un amministratore della macchina. Impostando un livello di trust che non consente l'accesso fisico al file system, se non eventualmente alle sole cartelle e file di nostro interesse (per es. minore o uguale a Medium), potremmo impedire simili attività, senza nemmeno richiedere un intervento di tipo sistemistico sul server, per configurare le ACL.

Ci limiteremmo a ridurre o "spegnere" l'accesso al file system.Se poi i cinque livelli di trust predefiniti non fanno al caso nostro, possiamo definire dei nostri file di configurazione personalizzati e configurarli nel machine.config nel modo seguente:

C#

```
<system.web>
    <securityPolicy>
        <trustLevel name="Full" policyFile="internal" />
        <trustLevel name="High" policyFile="web_
                                    hightrust.config" />
        <trustLevel name="Medium" policyFile="web_
                                  mediumtrust.config" />
        <trustLevel name="Low" policyFile="web_
                                    lowtrust.config" />
        <trustLevel name="Minimal" policyFile="web_
                                 minimaltrust.config" />
        <trustLevel name="Custom" policyFile="web_
                                  customtrust.config" />
    </securityPolicy>
    <trust level="Custom" originUrl="" />
</system.web>
```

I file di definizione dei livelli di trust non sono altro che file XML che elencano i permessi consentiti e i criteri in base ai quali sono consentiti. Il file web_customtrust.config può quindi essere creato a partire da uno di quelli già configurati, semplicemente personalizzandone il contenuto, aggiungendo o togliendo qualche permesso e condizione di assegnazione degli stessi.

Windows SharePoint Services 2003 per esempio sfrutta livelli di trust personalizzati, costruiti proprio in questo modo, utilizzando anche permessi personalizzati.

Quando abbiamo l'esigenza di eseguire minime porzioni di codice con un livello di trust maggiore, rispetto a quello standard e restrittivo da noi previsto, possiamo personalizzare il livello di trust, definendo per esempio un gruppo di permessi custom per gli assembly firmati con la nostra chiave privata, oppure possiamo anche inserire nella GAC gli assembly, rendendoli in questo modo dotati di Full Trust e dichiarando che consentiamo a livelli di trust inferiori di invocarli, utilizzando l'attributo AllowPartiallyTrustedCallers.

Conclusioni

Abbiamo potuto capire che il motore di ASP.NET è molto maturo dal punto di vista della sicurezza.

È di fondamentale importanza per scrivere applicazioni sicure e controllabili avere presente le dinamiche di impersonificazione e di esecuzione del codice associato ai Security Context, così come è altrettanto importante considerare i livelli di trust personalizzati e non eseguire tutto il codice come Full Trust.

Bibliografia e Riferimenti

[1] Autori vari - *" ASP.NET Security"*, Wrox Press, 2002

[2] Luca Regnicoli e Roberto Brunetti - *"ASP.NET Full Contact"*, Mondadori Informatica, 2003

[3] Fritz Onion - *"Essential ASP.NET"*, Addison-Wesley, 2003

[4] Keith Brown - *" The .NET Developer's Guide to Windows Security"*, Addison-Wesley, 2004

[5] Paolo Pialorsi - *" La sicurezza del codice managed"*, Visual Basic Journal n. 60, Nov/Dic 2004

[6] http://msdn.microsoft.com/aspnet/

[7] http://www.devleap.com/Default.aspx?IdCategoria=ASPNET

[8] http://msdn.microsoft.com/asp.net/using/understanding/security/default.aspx

[9] http://www.pluralsight.com/keith/book/html/book.html

[10] http://support.microsoft.com/default.aspx?scid=kb;en-us;329290

La validazione dell'input con ASP.NET

ASP.NET offre dei completi controlli lato server che gestiscono la validazione dell'input, rendendo non più necessaria la scrittura di codice personalizzato lato client e server. In questo articolo vedremo i vari controlli di questo tipo, e la loro applicazione pratica

di Marco Bellinaso

I compiti più comuni per le applicazioni web sono la consultazione e l'aggiornamento di un database, ossia la raccolta di informazioni. Vengono così utilizzate form e controlli textbox o di altro tipo che l'utente deve compilare e inviare al server premendo un pulsante Submit.
Nella quasi totalità dei casi è necessario un qualche livello di validazione dell'input, per fare in modo che nel database non vengano inseriti valori errati che possano causare errori.
I controlli necessari devono essere fatti direttamente sul lato client, in modo che il form non venga inviato e poi ripresentato in caso di errore, provocando così un round-trip inutile al server.
Devono però essere presenti anche sul lato server, come ulteriore conferma, perché un utente potrebbe in teoria salvare la pagina HTML, modificare il sorgente e usarla per inviare dati non validi. Infine altro codice deve effettivamente informare l'utente degli errori commessi nella compilazione della scheda.

Marco Bellinaso *lavora come trainer, consulente e sviluppatore di tool per programmatori per conto di Code Architects S.r.l. (www.codearchitects.com), società specializzata in .NET. Fa parte del Team di VB–2–The–Max (www.vb2themax.com) e si occupa con particolare interesse di soluzioni web distribuite con il framework .NET, sia con VB che C#. È coautore di quattro libri su .NET per Wrox Press, tra cui "ASP.NET Website Programming" (tradotto in italiano da HOEPLI) e "Fast Track ASP.NET". Può essere contattato tramite e–mail all'indirizzo mbellinaso@infomedia.it.*

Ciò che deriva da questa introduzione è che per fare una cosa semplice come la raccolta di dati bisogna scrivere parecchio codice per il controllo client e server, riempiendo le pagine di righe spesso uguali e andando comunque incontro a possibili errori se non si è abituati a lavorare con JavaScript (necessario per la compatibilità col client).
Qualcuno sicuramente starà obiettando che basta raggruppare le routine necessarie in file esterni da includere dove serve riutilizzando il codice già scritto, ma quanti sono in realtà riusciti ad ottenere delle routine veramente flessibili e un effettivo riuso che non costringa a personalizzare ogni volta qualcosa?
E comunque un po' di codice bisogna comunque sempre scriverlo per chiamare le routine di controllo per i vari campi.

ASP.NET viene fornito di serie con un gran numero di controlli lato server che aiutano il programmatore nei modi più disparati: creano tabelle, griglie avan-

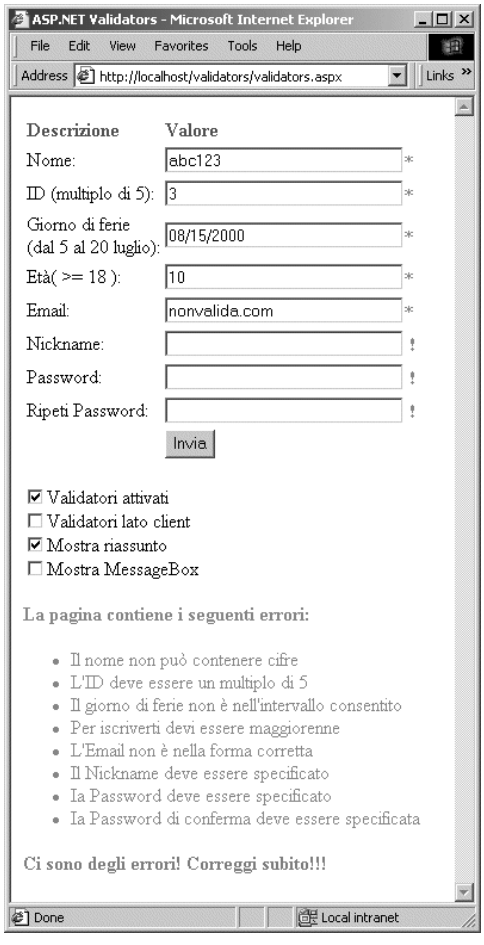


Figura 1 La pagina di esempio mostra il report degli errori

zate ed editabili, calendari, mostrano banner e molto altro.

Poiché la validazione dell'input è un compito così diffuso, un set di questi controlli si occupa proprio di questo, liberando il programmatore da questo noioso onere.

Tutto ciò che bisogna fare è dichiarare un nuovo controllo e senza scrivere una riga di codice per la validazione questa verrà effettuata sia sul client che sul server, controllando che i campi di input rispettino i vari tipi di regole definite dai diversi tipi di controllo.

Se poi il tipo di controllo che serve non è fornito di default, è allora possibile estenderli con una qualsiasi logica di validazione custom.

La classe base dei controlli validatori

Sono presenti controlli diversi, ognuno dei quali svolge un controllo diverso, ed è possibile sommarli per sommare i controlli effettuati sullo stesso campo.

Tutti questi componenti sono però derivati da una classe di base, *BaseValidator*, che espone le seguenti proprietà comuni:

- *ControlToValidate*: proprietà che indica il controllo il cui input deve essere validato;
- *Display*: proprietà che specifica come visualizzare il messaggio di errore.
  Se "static" lo spazio necessario al testo di notifica verrà calcolato e aggiunto alla pagina in anticipo, se "dynamic" lo spazio verrà ricavato in modo dinamico al momento di mostrare gli errori. Attenzione che sebbene la seconda possibilità possa sembrare attraente, in alcuni casi potrebbe cambiare di molto il layout della pagina, spaesando l'utente;
- *EnableClientScript*: proprietà booleana che indica se la validazione lato client deve essere effettuata o meno;
- *Enabled*: proprietà booleana che specifica se il controllo è attivo o meno, ossia se il campo di input associato verrà validato o no;
- *ErrorMessage*: stringa di errore che verrà mostrata nel sommario degli errori da un altro controllo, ValidationSummary;
- *ForeColor*: è il colore per il testo del messaggio che comparirà accanto o sotto un campo di input quando la sua validazione darà esito negativo. Di default è rosso;
- *IsValid*: proprietà booleana che indica se il contenuto del campo di input associato è valido o meno;
- *Validate*: metodo che compie la ri-validazione dell'input e aggiorna la proprietà IsValid di conseguenza.

La classe BaseValidator ha in realtà anche altre proprietà o metodi, come BackColor, Font e altri, ma sono tutti ereditati dalle classi Label e WebControl. BaseValidator è una classe astratta (dichiarata con *abstract* dal punto di vista dei programmatori C#, o *MustInherit* per chi usa VB.NET), ossia una classe che non può essere usata direttamente ma che può essere solo ereditata da altre classi, queste ultime usufruibili.

Tabella 1 I possibili valori per la proprietà Operator

Valore	Confronto tra il valore dell'input e quello di un altro controllo o valore costante
Equal	I due valori devono essere uguali
NotEqual	L'input deve essere diverso dal secondo valore
GreaterThan	L'input deve essere maggiore del secondo valore
GreaterThanEqual	L'input deve essere maggiore o uguale del secondo valore
LessThan	L'input deve essere minore del secondo valore
LessThanEqual	L'input deve essere minore o uguale del secondo valore
DataTypeCheck	Viene controllato solo che l'input sia del tipo specificato dalla proprietà Type

Il senso è che oltre ai membri di base, ogni classe derivata aggiunge delle nuove proprietà che cambiano le regole di validazione utilizzate.

Cominciamo a vedere controllo per controllo, presentando per ognuno un esempio.

I controlli per la validazione di base

Il controllo più semplice è *RequiredFieldValidator*, il cui unico scopo è verificare che il contenuto di una textbox non sia nullo (una serie di spazi e basta vengono considerati nulli).

L'unica proprietà aggiuntiva si chiama *InitialValue* ed esprime il valore iniziale dell'input, che se non cambiato verrà considerato come errore.

Spesso comunque viene usato nella sua forma più semplice:

> Con le regular expression si possono definire regole molto complesse

```
<asp:TextBox runat="server" Width="200px" ID="Name" />
<asp:RequiredFieldValidator runat="server"
                                        ID="ValidateName"
ControlToValidate="Name"
ErrorMessage= "Il Nome deve essere specificato"
Display="dynamic">*
</asp:RequiredFieldValidator>
```

Il campo "Name" è quello da controllare, il carattere * verrà mostrato accanto alla casella di testo in caso di errore alla pressione del pulsante Submit o anche quando il controllo perde il focus se si sta usando Internet Explorer 5 o superiore.

Contrariamente a quanto si potrebbe pensare ad intuito, la stringa per *ErrorMessage* non viene mostrata accanto al controllo, ma verrà usata da un altro controllo che vedremo tra breve.

Un'altra cosa da notare è che è stato specificato l'ID per il controllo: in realtà dare un nome al controllo è necessario solo quando si deve accedere ad esso tramite codice per cambiare o leggere le sue proprietà.

Il secondo controllo è il *RangeValidator*, e verifica che l'input sia compreso in un intervallo predefinito.

Il bello di questo controllo è che non controlla solo numeri interi, ma anche valori Double, String, Currency e Date.

Il tipo di dato viene specificato tramite la proprietà *Type*, mentre *MinimumValue* e *MaximumValue* definiscono l'intervallo. Il seguente esempio controlla che la data specificata sia compresa tra il 5 e il 20 agosto 2001:

```
<asp:TextBox runat="server" Width= "200px" ID="DayOff" />
<asp:RangeValidator runat= "server" ID="ValidateDayOff2"
ControlToValidate="DayOff"
MinimumValue="08/05/2001"
MaximumValue="08/20/2001"
Type="Date"
ErrorMessage="Il giorno di ferie non &egrave; nell'inter-
                                        vallo consentito"
Display="dynamic">*
</asp:RangeValidator>
```

Da notare che le date vengono specificate nel formato dettato dalla versione del sistema operativo installata, quindi mm/dd/aaaa nel caso di Windows 2000 inglese.

Un controllo simile a quello appena visto è *CompareValidator*, che confronta il contenuto di un controllo di input con un altro controllo o con un altro valore stabilito a priori.

> La convalida dell'input fa parte di quasi tutte le applicazioni web

È simile a RangeValidator perché ha la stessa proprietà Type che lo rende flessibile.

Nel caso si voglia validare l'input con il contenuto di un altro controllo, questo secondo controllo dovrà essere specificato dalla proprietà *ControlToCompare*, mentre si usa *ValueToCompare* se si vuole fare il confronto con un valore.

Il componente supporta diversi tipi di confronto, quello da effettuare deve essere specificato tramite la proprietà Operator, che accetta i valori specificati in **Tabella 1**.

Nell'esempio che segue viene controllato che l'età inserita sia di almeno 18 anni:

```
<asp:TextBox runat="server" Width="200px" ID="Age" />
<asp:CompareValidator runat="server" ID="ValidateAge"
ControlToValidate="Age"
ValueToCompare="18"
Type="Integer"
Operator="GreaterThanEqual"
ErrorMessage="Per iscriverti devi essere maggiorenne"
Display="dynamic">*
</asp:CompareValidator>
```

Mentre questo assicura che l'utente abbia specificato la stessa password in due campi textbox:

```
<asp:CompareValidator runat="server"
ControlToValidate="Password2"
ControlToCompare="Password"
Type="String"
ErrorMessage="La password di conferma non corrisponde"
Display="dynamic"><img src="imgError.gif" border="0">
</asp:CompareValidator>
```

Nell'ultimo esempio c'è un'altra cosa da notare: prima del tag di chiusura del controllo non c'è il solito asterisco degli esempi precedenti, ma un tag HTML <IMG> per mostrare un'immagine.

Il risultato è rappresentato in **Figura 1**, accanto alle ultime tre caselle di testo. Un altro controllo è *RegularExpressionValidator*, che verifica se l'input rispetta la regular expression specificata dalla proprietà *ValidationExpression*.

Questo controllo offre molta potenza, perché le espressioni possono essere anche molto complicate e si adattano a molti tipi di verifiche.

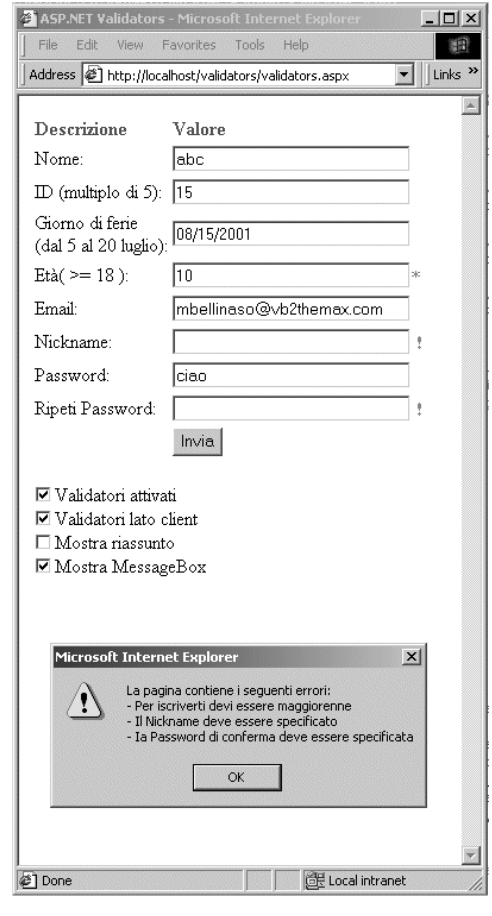


Figura 2 La pagina di esempio mostra la lista degli errori in una message box

Listato 1 Il codice completo della pagina di esempio (*continua*)

```
<%@ Page Language="C#" %>

<html>
    <head>
        <title>ASP.NET Validators</title>

        <script language="JavaScript">
            function EmpIDClientValidate(ctl, args)
            {
                args.IsValid=(args.Value%5 == 0);
            }
        </script>
    </head>

    <body>

    <form method=post runat="server" ID="FormValidators">
    <asp:table runat="server">
        <asp:TableRow>
            <asp:TableCell Width="50px" Text="Descrizione"
                Font-Bold="true" ForeColor="blue" />
            <asp:TableCell Width="200px" Text="Valore"
                Font-Bold="true" ForeColor="blue" />
        </asp:TableRow>

        <asp:TableRow>
            <asp:TableCell Text="Nome:"/>
            <asp:TableCell>
                <asp:TextBox runat="server" Width="200px" ID="Name" />
                <asp:RequiredFieldValidator runat="server" ID="ValidateName"
                    ControlToValidate="Name"
                    ErrorMessage="Il Nome deve essere specificato"
                    Display="dynamic">*
                </asp:RequiredFieldValidator>
                <asp:RegularExpressionValidator runat="server" ID="ValidateName2"
                    ControlToValidate="Name"
                    validationExpression="[a-z A-Z]*"
                    ErrorMessage="Il nome non pu&ograve; contenere cifre"
                    Display="dynamic">*
                </asp:RegularExpressionValidator>
            </asp:TableCell>
        </asp:TableRow>

        <asp:TableRow>
            <asp:TableCell Text="ID (multiplo di 5):"/>
            <asp:TableCell>
                <asp:TextBox runat="server" Width="200px" ID="EmpID" />
                <asp:RequiredFieldValidator runat="server" ID="ValidateEmpID"
                    ControlToValidate="EmpID"
                    ErrorMessage="L'ID deve essere specificato"
                    Display="dynamic">*
                </asp:RequiredFieldValidator>
                <asp:CustomValidator runat="server" ID="ValidateEmpID2"
                    ControlToValidate="EmpID"
                    ClientValidationFunction="EmpIDClientValidate"
                    OnServerValidate="EmpIDServerValidate"
                    ErrorMessage="L'ID deve essere un multiplo di 5"
                    Display="dynamic">*
                </asp:CustomValidator>
```

Listato 1 (*segue*) Il codice completo della pagina di esempio

```
    </asp:TableCell>
</asp:TableRow>

<asp:TableRow>
    <asp:TableCell Text="Giorno di ferie <br>(dal 5 al 20 luglio):"/>
    <asp:TableCell>
        <asp:TextBox runat="server" Width="200px" ID="DayOff" />
        <asp:RequiredFieldValidator runat="server" ID="ValidateDayOff"
            ControlToValidate="DayOff"
            ErrorMessage="Il giorno di ferie deve essere specificato"
            Display="dynamic">*
        </asp:RequiredFieldValidator>
        <asp:RangeValidator runat="server" ID="ValidateDayOff2"
            ControlToValidate="DayOff"
            MinimumValue="08/05/2001"
            MaximumValue="08/20/2001"
            Type="Date"
            ErrorMessage="Il giorno di ferie non &egrave; nell'intervallo consentito"
            Display="dynamic">*
        </asp:RangeValidator>
    </asp:TableCell>
</asp:TableRow>

<asp:TableRow>
    <asp:TableCell Text="Et&agrave;( >= 18 ):"/>
    <asp:TableCell>
        <asp:TextBox runat="server" Width="200px" ID="Age" />
        <asp:RequiredFieldValidator runat="server"
            ControlToValidate="Age"
            ErrorMessage="L'et&agrave; deve essere specificata"
            Display="dynamic">*
        </asp:RequiredFieldValidator>
        <asp:CompareValidator runat="server" ID="ValidateAge"
            ControlToValidate="Age"
            ValueToCompare="18"
            Type="Integer"
            Operator="GreaterThanEqual"
            ErrorMessage="Per iscriverti devi essere maggiorenne"
            Display="dynamic">*
        </asp:CompareValidator>
    </asp:TableCell>
</asp:TableRow>

<asp:TableRow>
    <asp:TableCell Text="Email:"/>
    <asp:TableCell>
        <asp:TextBox runat="server" Width="200px" ID="Email" />
        <asp:RequiredFieldValidator runat="server"
            ControlToValidate="Email"
            ErrorMessage="L'Email deve essere specificata"
            Display="dynamic">*
        </asp:RequiredFieldValidator>
        <asp:RegularExpressionValidator runat="server" ID="ValidateEmail"
            ControlToValidate="Email"
            validationExpression=".*@.{2,}\..{2,}"
            ErrorMessage="L'Email non &egrave; nella forma corretta"
            Display="dynamic">*
        </asp:RegularExpressionValidator>
    </asp:TableCell>
</asp:TableRow>
```

Listato 1 (*segue*) Il codice completo della pagina di esempio

```
        <asp:TableRow>
            <asp:TableCell Text="Nickname:"/>
            <asp:TableCell>
                <asp:TextBox runat="server" Width="200px" ID="Nick" />
                <asp:RequiredFieldValidator runat="server" ID="ValidateNick"
                    ErrorMessage="Il Nickname deve essere specificato"
                    ControlToValidate="Nick"
                    Display="dynamic"><img src="imgError.gif" border="0">
                </asp:RequiredFieldValidator>
            </asp:TableCell>
        </asp:TableRow>

        <asp:TableRow>
            <asp:TableCell Text="Password:"/>
            <asp:TableCell>
                <asp:TextBox runat="server" Width="200px" ID="Password" />
                <asp:RequiredFieldValidator runat="server"
                    ControlToValidate="Password"
                    ErrorMessage="Ia Password deve essere specificato"
                    Display="dynamic"><img src="imgError.gif" border="0">
                </asp:RequiredFieldValidator>
            </asp:TableCell>
        </asp:TableRow>

        <asp:TableRow>
            <asp:TableCell Text="Ripeti Password:"/>
            <asp:TableCell>
                <asp:TextBox runat="server" Width="200px" ID="Password2" />
                <asp:RequiredFieldValidator runat="server"
                    ControlToValidate="Password2"
                    ErrorMessage="Ia Password di conferma deve essere specificata"
                    Display="dynamic"><img src="imgError.gif" border="0">
                </asp:RequiredFieldValidator>
                <asp:CompareValidator runat="server"
                    ControlToValidate="Password2"
                    ControlToCompare="Password"
                    Type="String"
                    ErrorMessage="La password di conferma non corrisponde"
                    Display="dynamic"><img src="imgError.gif" border="0">
                </asp:CompareValidator>
            </asp:TableCell>
        </asp:TableRow>

        <asp:TableRow>
            <asp:TableCell />
            <asp:TableCell>
                <asp:Button runat="server" Text="Invia" ID="Submit" OnClick="Submit_Click" />
            </asp:TableCell>
        </asp:TableRow>
    </asp:Table>

    <br>
    <asp:CheckBox runat="server" ID="EnableValidators"
        Checked="true" AutoPostBack="true"/>Validatori attivati
    <br>
    <asp:CheckBox runat="server" ID="EnableClientSide"
        Checked="true" AutoPostBack="true"/>Validatori lato client
    <br>
    <asp:CheckBox runat="server" ID="ShowSummary"
        Checked="true" AutoPostBack="true"/>Mostra riassunto
```

Listato 1 (*segue*) Il codice completo della pagina di esempio

```
    <br>
    <asp:CheckBox runat="server" ID="ShowMsgBox"
        Checked="false" AutoPostBack="true"/>Mostra MessageBox
    <br><br>

    <asp:ValidationSummary runat="server" ID="ValidationSum"
        DisplayMode="BulletList"
        HeaderText="<b>La pagina contiene i seguenti errori:</b>"
        ShowSummary="true"
    />

    </form>

    <asp:Label runat="server" ID="Result" ForeColor="magenta" Font-Bold="true"/>

    </body>
</html>

<script language="C#" runat="server">

void Page_Load()
{
    Result.Text="";
    foreach (BaseValidator valCtl in Page.Validators)
    {
        valCtl.Enabled=EnableValidators.Checked;
        valCtl.EnableClientScript=EnableClientSide.Checked;
    }
    ValidationSum.ShowMessageBox=ShowMsgBox.Checked;
    ValidationSum.ShowSummary=ShowSummary.Checked;
}

void EmpIDServerValidate(object sender, ServerValidateEventArgs args)
{
    try
    {
        args.IsValid = (int.Parse(args.Value)%5 == 0);
    }
    catch
    {
        args.IsValid = false;
    }
}

void Submit_Click(object sender, EventArgs e)
{
    if (Page.IsValid)
        Result.Text="Grazie per aver inviato i tuoi dati";
    else
        Result.Text="Ci sono degli errori! Correggi subito!!!";
}

</script>
```

Per esempio di seguito è dichiarato un controllo che verifica che il campo Nome contenga solo lettere minuscole o maiuscole o spazi, ma niente cifre o altri caratteri:

```
<asp:RegularExpressionValidator runat="server"
                                    ID="ValidateName2"
ControlToValidate="Name"
validationExpression="[a-z A-Z]*"
ErrorMessage="Il nome non pu&ograve; contenere cifre"
Display="dynamic">*
</asp:RegularExpressionValidator>
```

mentre l'espressione che segue controlla che il campo Email contenga un indirizzo dal formato corretto (alcuni caratteri)

```
…
validationExpression=".*@.{2,}\..{2,}"
…
```

Listato 2 Alcune routine del file WebUIValidation.js per la validazione lato client

```
function ValidatorCompare(operand1, operand2, operator, val) {
    var dataType = val.type;
    var op1, op2;
    if ((op1 = ValidatorConvert(operand1, dataType, val)) == null)
        return false;
    if (operator == "DataTypeCheck")
        return true;
    if ((op2 = ValidatorConvert(operand2, dataType, val)) == null)
        return true;
    switch (operator) {
        case "NotEqual":
            return (op1 != op2);
        case "GreaterThan":
            return (op1 > op2);
        case "GreaterThanEqual":
            return (op1 >= op2);
        case "LessThan":
            return (op1 < op2);
        case "LessThanEqual":
            return (op1 <= op2);
        default:
            return (op1 == op2);
    }
}

function CompareValidatorEvaluateIsValid(val) {
    var value = ValidatorGetValue(val.controltovalidate);
    if (value == "")
        return true;
    var compareTo = "";
    if (null == document.all[val.controltocompare]) {
        if (typeof(val.valuetocompare) == "string") {
            compareTo = val.valuetocompare;
        }
    }
    else {
        compareTo = ValidatorGetValue(val.controltocompare);
    }
    return ValidatorCompare(value, compareTo, val.operator, val);
}

function RegularExpressionValidatorEvaluateIsValid(val) {
    var value = ValidatorGetValue(val.controltovalidate);
    if (value == "")
        return true;
    var rx = new RegExp(val.validationexpression);
    var matches = rx.exec(value);
    return (matches != null && value == matches[0]);
}
```

Questa espressione impone che ci sia un numero di caratteri qualsiasi (".*") seguiti da una '@', almeno due caratteri (".{2,}"), un punto e almeno altri due caratteri. In realtà questa espressione non è perfetta perché l'indirizzo è considerato valido anche se si inseriscono caratteri invalidi come ^ o altri.

È comunque possibile ottenere un'espressione molto più completa, per maggiori informazioni riguardo la sintassi da usare potete consultare [1].

Un controllo per la validazione personalizzata

I tre controlli visti finora probabilmente rendono molto soddisfatti la maggior parte dei programmatori ASP, ma non è tutto qui. C'è un altro controllo, *CustomValidator*, al quale possono essere associate due routine personalizzate di verifica, per controllare l'input secondo una qualsiasi logica.

Una procedura serve per la verifica lato client (opzionale), l'altra lato server, e vengono associate tramite la proprietà *ClientValidationFunction* e il gestore di evento *OnServerValidate*, rispettivamente.

Questo controllo è utile quando si deve convalidare un valore che deve seguire delle regole particolari, ad esempio un numero di carta di credito.

Iniziamo vedendo come deve essere dichiarato il controllo:

```
<asp:TextBox runat="server" Width="200px"
                                    ID="EmpID" />
```

```
<asp:CustomValidator runat="server" ID="ValidateEmpID2"
ControlToValidate="EmpID"
ClientValidationFunction="EmpIDClientValidate"
OnServerValidate="EmpIDServerValidate"
ErrorMessage="L'ID deve essere un multiplo di 5"
Display="dynamic">*
</asp:CustomValidator>
```

La due funzioni di controllo sono dichiarate nello stesso modo, salvo ovviamente che una è in JavaScript (per ottenere la massima compatibilità con i browser) e la seconda in C#.

Hanno due parametri, il primo è un oggetto che fa riferimento al controllo validatore chiamante, e il secondo è un oggetto contenente l'attuale valore del controllo di input (proprietà Value) e una proprietà, IsValid, il cui valore verrà impostato come risultato della verifica.

Nell'esempio preparato le due funzioni controllano che il numero specificato nell'input sia un multiplo di 5.

Ecco la versione lato client:

```
<script language="JavaScript">
function EmpIDClientValidate(ctl, args)
{
args.IsValid=(args.Value%5 == 0);
}
</script>
```

Ed ecco quella molto simile in versione C# per il lato server:

```
void EmpIDServerValidate(object sender,
                         ServerValidateEventArgs args)
{
    try
    {
      args.IsValid = (int.Parse(args.Value)%5 == 0);
    }
    catch
    {
      args.IsValid = false;
    }
}
```

Un resoconto degli errori

L'ultimo controllo della serie è dedicato alla creazione di un rapporto sugli errori incontrati al momento dell'invio dei dati al server.

Questo controllo si chiama *ValidationSummary*, e mostra i messaggi di errore (quelli specificati dalla proprietà ErrorMessage) di tutti gli altri validatori la cui proprietà IsValid è false.

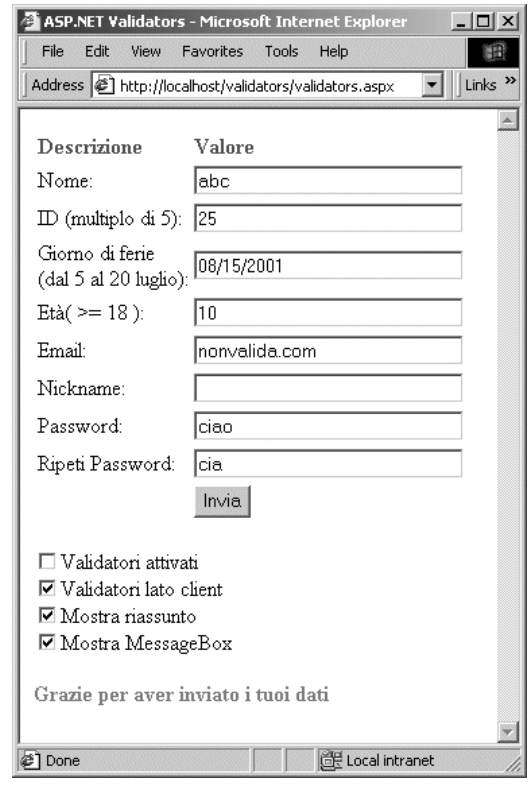


Figura 3 I validatori sono disabilitati, quindi l'input non viene controllato e la pagina viene inviata con successo

Le proprietà caratteristiche di questo componente sono:

- *HeaderText*: il titolo del report;
- *DisplayMode*: lo stile con il quale saranno visualizzati tutti gli errori.
  Le possibili scelte sono *SingleParagraph*, che mostra tutti i messaggi sulla stessa riga e separati solo da uno spazio, List che ne mostra uno per riga o BulletList che ne mostra uno per riga con un pallino accanto (come con il tag <LI>);
- *ShowSummary*: se true, il report viene mostrato direttamente nella pagina con lo stile indicato dalla proprietà precedente;
- *ShowMessageBox*: se vale true e se è attiva la validazione client side (proprietà EnableClientScript, true di default) gli errori vengono mostrati tramite una message box, una di quelle che si ottengono con Alert di JavaScript e raffigurata in **Figura 2**.

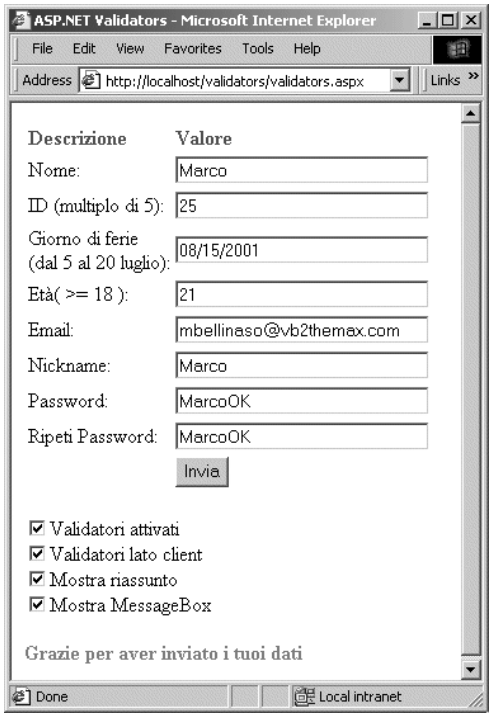


Figura 4 Un esempio di input corretto

Questa proprietà non esclude la precedente, entrambe possono essere true e gli errori verranno mostrati in entrambi i modi. Ecco un esempio per la dichiarazione:

```
<asp:ValidationSummary runat="server" ID="ValidationSum"
DisplayMode="BulletList"
HeaderText="<b>La pagina contiene i seguenti errori:</b>"
ShowSummary="true"
/>
```

Una sola nota particolare su questo controllo: il report nella pagina viene visualizzato anche se il controllo è stato dichiarato all'esterno del blocco di tag <form>…</form>, ma se si vogliono mostrare gli errori tramite message box allora deve essere necessariamente all'interno.

Modificare le proprietà a runtime

Una delle cose più belle di ASP.NET è che a runtime, all'interno delle procedure di evento, è possibile accedere a tutti i controlli ai quali era stato assegnato un ID, per leggere o modificare qualsiasi loro proprietà modificando di conseguenza il loro comportamento.

Nella pagina di esempio ci sono delle checkbox che permettono di abilitare o meno la validazione totale, solo quella client side, l'apertura della message box e il report nella pagina. Quando una di queste checkbox viene cliccata, l'evento causa un *postback* della pagina (per farlo il controllo deve essere dichiarato con la proprietà AutoPostBack uguale a true), ossia la pagina viene inviata a se stessa e ricaricata.

Nell'evento Page_Load i nuovi valori vengono letti e le proprietà cambiate di conseguenza. Ciò che è interessante è la possibilità di accedere a tutti i controlli con un solo ciclo grazie al costrutto *foreach* di C# (o *for each* in VB.NET), quindi senza la necessità di specificare e conoscere gli ID di ciascun componente.

Ecco l'esempio:

```
foreach (BaseValidator valCtlin Page.Validators)
{
valCtl.Enabled=EnableValidators.Checked;
valCtl.EnableClientScript=EnableClientSide.Checked;
}
```

Le altre due proprietà si riferiscono invece al solo controllo ValidationSummary:

```
ValidationSum.ShowMessageBox=ShowMsgBox.Checked;
ValidationSum.ShowSummary=ShowSummary.Checked;
```

> Ciò che viene prodotto per il client è semplice HTML e JavaScript

In **Figura 3** si vede che quando i validatori sono disabilitati viene mostrato il messaggio di conferma anche se l'input è in gran parte invalido, mentre in **Figura 4** si vede la stessa schermata ma con i validatori attivati e l'input completamente corretto.

Ultima domanda è come venga mostrato quel messaggio.

Quando il pulsante "Invia" viene premuto esso genera un evento Click che può essere gestito

per controllare la proprietà IsValid della pagina: il valore sarà true solo se sarà tale la proprietà IsValid di tutti i validatori sulla pagina.

Da notare che se è attivato il controllo lato client e se il browser è di alto livello con il supporto DHTML questo evento non verrà sollevato finché la pagina non viene inviata correttamente.

Per vedere il messaggio di errore generato sul server bisogna necessariamente disabilitare gli script sul client, come da **Figura 1**.

Nel **Listato 1** (disponibile sul sito ftp di Infomedia) trovate tutto il codice della pagina, compresa quest'ultima parte sulla programmazione server side.

Infine, ricordate che la validazione dell'input avviene ogni volta che si tenta di fare un postback della pagina, e questo accade anche quando si preme un altro bottone, ad esempio un eventuale Cancella o Elimina Record.

Per questi pulsanti ovviamente non si vuole che l'input venga controllato, e allora nella loro dichiarazione basta impostare la proprietà *CausesValidation* su false.

Un'occhiata dietro le quinte

Prima di concludere è interessante vedere come funziona veramente il procedimento di validazione, ossia qual è il trucco per rendere automatico il tutto.

La validazione viene effettuata anche sul client, quindi vuol dire che viene prodotto normale codice JavaScript per il controllo di tutti i valori.

Dando un'occhiata al codice HTML prodotto per la pagina mostrata si vede che all'inizio viene incluso il seguente file .JS:

```
<script language="javascript"
src="/aspnet_client/system_web/1_0_2914_16/
                                    WebUIValidation.js">
</script>
```

Nel seguito viene definito un array con tutti i controlli validatori, come segue:

```
var Page_Validators =  new Array(
document.all["ValidateName"],
document.all["ValidateName2"],
document.all["ValidateEmpID"],
document.all["ValidateEmpID2"], …etc…)
```

e poco dopo c'è una chiamata a ValidatorOnLoad, una funzione che si trova all'interno del file JavaScript incluso, la quale usa un ciclo per processare ogni elemento dell'array controllandolo ed eventualmente associandolo a delle procedure di evento, tutto tramite routine secondarie.

Tanto per farsi un'idea, nel **Listato 2** sono riportate le funzioni che implementano i controlli CompareValidator e RegularExpressionValidator.

> I controlli sono oggetti ed è possibile cambiare le loro proprietà a runtime

Infine, le variabili globali Page_IsValid e Page_BlockSubmit indicano se la pagina è risultata valida o se l'invio al server deve essere bloccato.

Conclusioni

Sebbene ASP.NET non faccia cose veramente magiche, tutto quello che arriva al browser è semplice HTML e codice di scripting client side, è quasi magia poter creare applicazioni significative in molto meno tempo rispetto a quello che occorrerebbe con l'ASP classico.

Il set di componenti di validazione fanno la loro bella parte come si è visto, tanto più che verranno usati in modo più o meno estensivo in praticamente tutte le applicazioni.

Oltre alla semplicità della dichiarazione, alla possibilità di personalizzarli e cambiare il loro comportamento a runtime con poche righe, riducendo il tempo di sviluppo, viene ridotto quasi a zero anche il tempo di debug per quest'aspetto delle applicazioni, infatti il codice JavaScript prodotto – spesso fonte di errori per i programmatori ASP non abituati alla programmazione lato client – sarà stato sicuramente testato in lungo e in largo. E pensare che tutto ciò è solo un millesimo della potenza di ASP.NET.

Bibliografia

[1] M. Merlano – "Utilizzare le regular expression", VBJ n. 33 Mag/Giu 2000, Gruppo Editoriale Infomedia.

NOVITÀ

euro 5,00

Un testo pratico e semplice per l'apprendimento di C#, il nuovo e rivoluzionario linguaggio di programmazione, concepito da Microsoft, per l'ambiente .NET. Con spiegazioni abilmente concepite, suggerimenti nati dalla profonda esperienza accumulata negli anni, ed esempi semplici ed efficaci, Baccan prende in esame gli aspetti salienti di C#, sia della versione 1.1 che della **nuovissima versione 2.0.**

corso di C#

Questo testo è l'evoluzione, riveduta e potenziata, del corso a puntate tenuto per rivista DEV di Infomedia. Sono state riscritte per intero delle parti, per renderle attuali, è stato fatto un lavoro di ricontrollo di tutti gli esempi e soprattutto è stata inserita la spiegazione di tutti i nuovi costrutti di C# 2.0.

pagg. 96
ISBN 8881500167
€ 5.00

GRUPPO EDITORIALE INFOMEDIA
THE SOFTWARE SIDE OF YOUR MIND

Tel. 0587/736460
e-mail:book@infomedia.it
WEB: http://book.infomedia.it

I design pattern più famosi

implementati in VB .NET

Seconda puntata

Il pattern Observer consente di notificare ad un insieme di 'osservatori' le modifiche apportate ad un oggetto

di Lorenzo Vandoni

Nel primo articolo di questa serie è stata presentata un'introduzione al movimento dei pattern, ed è stato fornito un esempio di implementazione VB.NET di uno tra i pattern più famosi tra quelli presentati nel libro "Design Patterns" di Gamma, Helm, Johnson e Vlissides. In questa puntata verrà preso in considerazione un altro dei pattern presentati in questo testo, denominato Observer. La descrizione di questo pattern verrà effettuata utilizzando la convenzione presentata nella prima puntata, costituita da un insieme di proprietà associate ad una descrizione testuale che permettono di presentare separatamente il problema, i vincoli e la soluzione del pattern. Anche in questo caso, infine, verrà fornito un esempio di implementazione VB.NET.

Il pattern Observer, descrizione

La descrizione del pattern è costituita dai paragrafi seguenti, che ne evidenziano lo scopo, le motivazioni, i problemi affrontati e la soluzione suggerita.

- *Intent*. Observer è un pattern che permette di associare ad un oggetto un insieme di *osservatori*, a cui viene inviata una notifica ogni volta che all'oggetto vengono apportate delle modifiche.
- *Motivation*. Il classico esempio che viene fatto per spiegare il problema che il pattern intende risolvere è quello di un'applicazione a documenti multipli, in cui ci sono diverse finestre aperte che mostrano viste differenti degli stessi dati. Ad esempio, una finestra potrebbe mostrare i dati delle vendite in una griglia, mentre un'altra finestra potrebbe mostrare gli stessi dati in un grafico a torta. In casi del genere, tipicamente, si vuole fare in modo che le modifiche apportate in una delle finestre (ad esempio, la selezione di un mese diverso) vengano automaticamente rispecchiate dalle altre. Questa necessità può però portare ad una progettazione sbagliata della struttura del programma, in cui ogni classe risulta a conoscenza delle altre, pregiudicando così sia la riusabilità delle stesse, sia la possibilità di aggiungere nuove classi, ovvero nuovi tipi di viste sui dati. La soluzione proposta dal pattern è quella di distinguere gli oggetti osservatori, interessati a rispecchiare immediatamente le modifiche apportate, dagli

Lorenzo Vandoni *è laureato in Informatica, ed è uno specialista di progettazione e sviluppo con tecniche e linguaggi object–oriented. Ha collaborato alla realizzazione di framework, strumenti di sviluppo e software commerciali in C++, Java e Visual Basic. Può essere contattato tramite e–mail all'indirizzo vandoni@infomedia.it.*

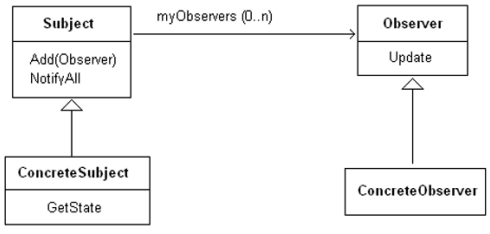


Figura 1 Architettura del pattern Observer

oggetti osservati, quelli cioè su cui tali modifiche vengono effettuate. Nel seguito dell'articolo supporremo, per semplicità, che esista un unico oggetto osservato ed uno o più osservatori, ma la soluzione che vedremo si può applicare anche ai casi più complessi in cui gli oggetti osservati siano più d'uno. Questo tipo di interazione è anche nota col termine publish-subscribe. L'oggetto osservato, cioè, può essere assimilato a un fornitore di servizi, mentre gli osservatori possono essere visti come i sottoscrittori di una sorta di abbonamento. Il sottoscrittore avvisa il publisher che è interessato a ricevere informazioni aggiornate ogni volta che al publisher succede qualcosa, ma è poi compito di quest'ultimo avvisare i sottoscrittori ogni qual volta si verifica qualcosa di interessante.

- *Applicabilità*. Il pattern è applicabile in tutti quei contesti in cui le modifiche apportate ad alcuni oggetti o dati dell'applicazione devono essere note ad altre parti dell'applicazione, soprattutto se il numero di oggetti o dati variati, e il numero di sezioni dell'applicazione interessate ad essere informate di tali modifiche non sono noti a priori o possono variare nel tempo.

Soluzione del problema

La soluzione del problema è costituita dal diagramma di classi mostrato in **Figura 1**. I paragrafi seguenti ne forniscono una descrizione più dettagliata.

- Structure. Il cuore della soluzione è costituito dalle due interfacce *Subject* e *Observer*, che rappresentano un generico oggetto osservato e un generico osservatore. Ogni classe, le cui istanze dovranno essere gli oggetti osservati, dovrà implementare la prima interfaccia, mentre ogni classe usata per implementare un osservatore dovrà implementare la seconda. Nel diagramma, queste classi sono rappresentate da *ConcreteSubject* e *ConcreteObserver*, ma in un caso reale potranno esistere molte implementazioni per ognuna delle due interfacce.

- Participants. Vengono di seguito descritte le responsabilità delle interfacce e delle classi precedentemente introdotte:

  - l'interfaccia Subject mentiene una lista di Observer, e fornisce un metodo Add tramite il quale un Observer può registrarsi. Il metodo NotifyAll serve per notificare a tutti gli Observer che si sono registrati, e che sono mantenuti nella lista, un'avvenuto cambiamento di stato.
  - la classe ConcreteSubject fornisce un'implementazione dell'interfaccia, e implementa un metodo GetState che permette agli Observer di conoscere lo stato attuale dell'oggetto osservato.
  - l'interfaccia Observer contiene un metodo Update che viene utilizzato da parte del metodo NotifyAll del subject per informare i singoli Observer di un avvenuto cambiamento di stato.
  - la classe ConcreteObserver implementa l'interfaccia e ha la responsabilità di effettuare un aggiornamento della visualizzazione, richiamando il metodo GetState del ConcreteSubject osservato, per conoscerne lo stato attuale.

- Collaborations. La collaborazione tra gli oggetti che fanno parte del sistema è di fatto definita dalle responsabilità dei partecipanti, ma può essere utile chiarire questo aspetto mostrando la sequenza temporale di tale interazione:

  - come prima cosa, due oggetti Observer A e B notificano il proprio interesse ad essere informati dei cambiamenti di stato ad un Subject S utilizzando il metodo Add.
  - nel momento in cui avviene un cambiamento di stato, ad esempio a seguito di un'operazione dell'utente, il Subject utilizza il proprio metodo NotifyAll per notificare questo cambiamento ai vari Observer. Questo metodo, di fatto, richiama il meto-

do Update dei due Observer.
- ognuno dei due Observer mantiene un riferimento al Subject a cui è interessato, e a cui ha inviato il messaggio Add, e una volta che gli è stato notificato un cambiamento di stato, utilizza il metodo GetState di quest'ultimo per conoscerne lo stato attuale.
- una volta conosciuto lo stato attuale, ognuno degli Observer aggiorna il proprio stato o la propria interfaccia grafica, per rispecchiare il nuovo stato del Subject.

Conseguenze e benefici

I paragrafi seguenti descrivono i punti di forza della soluzione e mostrano alcune sue possibili applicazioni pratiche:

- Consequences. Il principale vantaggio che si ottiene è quello di svincolare totalmente il Subject dagli Observer che lo utilizzano. In questo modo si possono creare una o più classi ConcreteSubject che non devono conoscere nulla degli oggetti che verranno utilizzati per la loro visualizzazione, e saranno così facilmente riusabili in diverse applicazioni.

- Known uses. Questo pattern è utilizzato in molte applicazioni e librerie. Lo stesso modello asincrono di gestione degli eventi, comune a tutte le applicazioni Windows, può essere considerato come un'applicazione del pattern, in quanto l'oggetto che attiva l'evento può non sapere nulla degli oggetti che andranno a gestirlo. Personalmente ho applicato il pattern in vari contesti, uno dei quali, semplificato, costituisce l'esempio allegato a questo articolo.

Listato 1 Implementazione delle interfacce del pattern in VB.NET

```
''''''''''''''''''''''''''''
''' Interfaccia Observer '''
''''''''''''''''''''''''''''
Public Interface Observer
    Sub Update()
End Interface

''''''''''''''''''''''''''''
''' Interfaccia Subject '''
''''''''''''''''''''''''''''
Public Interface Subject
    Sub Add(ByVal oObs As Observer)
    Sub NotifyAll()
End Interface
```

Implementazione del pattern in VB.NET

L'esempio fornito è costituito da un'applicazione web, in cui è presente un piccolo banner, implementato tramite uno user control, che permette all'utente di selezionare la lingua utilizzata. Si intende costruire questo banner in modo che possa essere riusato in più applicazioni. Ovviamente, non è possibile prevedere a priori quali saranno, in queste applicazioni, gli elementi di interfaccia che dovranno variare a seconda della lingua selezionata, e in che modo. Questo banner è quindi un ottimo candidato per diventare un *Subject* del pattern.

Nell'esempio, le due interfacce, che si possono vedere anche nel **Listato 1**, sono definite all'interno del file *Interfaces.vb*.

L'esempio è costituito da una pagina principale, *WebForm1.aspx*, al cui interno vengono posizionati, con l'aiuto di una tabella, tre user control, denominati rispettivamente *choose_language.ascx*, *left_menu.ascx* e *page_body.ascx*. Il control *choose_language.ascx* implementa l'interfaccia *Subject*. Per farlo definisce una variabile di istanza *myObservers*, dove mantenere la lista di tutti gli *Observer*, e implementa i due metodi *Add* e *NotifyAll*.

I due control *left_menu.ascx* e *page_body.ascx* implementano l'interfaccia *Observer*, fornendo un metodo *Update*, che viene utilizzato per aggiornare il contenuto del control stesso.

All'interno di questo metodo lo stato del *Subject* viene interrogato in maniera indiretta, utilizzando il contenuto di una variabile di sessione. Il compito di assegnare al *Subject* l'elenco degli *Observer* viene implementato all'interno dell'evento *Page_Load* della pagina principale. Si può facilmente notare, dal codice, come il componente *choose_language.ascx* risulti del tutto indipendente dagli altri. Risulterà facile, quindi, riutilizzarlo all'interno di altre applicazioni.

Conclusioni

Observer è un pattern piuttosto semplice ma, come spesso capita coi pattern, spesso risulta difficile riconoscere nel proprio problema applicativo un caso in cui esso sia applicabile. Il grosso vantaggio di *Observer* è quello di consentire di creare un *Subject* riusabile, con evidenti vantaggi anche in fase di manutenzione. Il mancato utilizzo del pattern può provocare problemi a lungo termine, in quanto spesso porterà a una progettazione in cui i vari componenti dell'applicazione risulteranno strettamente connessi tra loro.

Implementazione della logica applicativa in programmi VB.NET

Seconda puntata

Logica 2

Mostriamo l'implementazione di alcune classi che consentono di incapsulare la logica applicativa

di Lorenzo Vandoni

Proseguiamo in questo articolo la discussione iniziata in [1], dove era stata mostrata una possibile soluzione per la realizzazione di classi .NET che implementino la logica applicativa del programma. La puntata precedente è stata dedicata al trattamento delle problematiche progettuali. In questo articolo ci dedicheremo invece ai problemi legati all'implementazione.

> L'idea è quella di utilizzare la classe DataRow di ADO.NET come base da cui ereditare un insieme di classi applicative

Il progetto da realizzare

Le considerazioni riportate in [1] ci hanno portato alla soluzione descritta in **Figura 1**. Questo diagramma mostra come strutturare un insieme di classi .NET che implementano la logica applicativa di un programma, evitando di scrivere codice ripetitivo e ridondante.

L'idea è quella di utilizzare la classe *DataRow* di ADO.NET come base da cui ereditare un insieme di classi applicative, come *Automobile* o *Fattura*. Tali classi saranno implementate solo quando realmente necessarie, scegliendo di utilizzare semplici istanze di *DataRow* ogni qualvolta la derivazione di una classe specifica si dimostrerà inutile.

Per ognuna di queste classi applicative verrà inoltre realizzata una classe derivata da *DataSet*, che avrà il compito di incapsulare uno specifico *DataAdapter*. Quest'ultimo implementerà la logica di mapping necessaria per creare, a partire dai dati del DBMS, un'istanza della classe applicativa ed eventuali istanze di classi accessorie, come ad esempio *Carrozzeria* e *Motore*.

Questa struttura risulta fortemente dipendente dall'architettura di ADO.NET, e quindi difficil-

Lorenzo Vandoni *è laureato in Informatica, ed è uno specialista di progettazione e sviluppo con tecniche e linguaggi object–oriented. Ha collaborato alla realizzazione di framework, strumenti di sviluppo e software commerciali in C++, Java e Visual Basic. Può essere contattato tramite e–mail all'indirizzo vandoni@infomedia.it.*

mente portabile su altri sistemi. Proprio per questo motivo, però, permette di soddisfare il principale obiettivo che ci eravamo inizialmente posti, ovvero quello di riuscire ad incapsulare la logica applicativa in uno specifico strato di software intermedio, senza per questo provocare la creazione di un insieme di classi banali e strutturalmente identiche tra loro. In tutti i casi in cui l'oggetto applicativo risulterà strutturalmente semplice, infatti, sarà sufficiente continuare ad utilizzare oggetti *DataSet* e *DataRow*, senza implementare alcuna classe derivata.

DataSet tipizzati

Per implementare le classi *Automobile* e *AutomobileSet* ci avvarremo di una importante caratteristica di Visual Studio, ovvero della possibilità di generare automaticamente il codice di un *DataSet* tipizzato.

Un *DataSet* tipizzato non è altro che una sottoclasse di *DataSet*, con membri pubblici aggiuntivi che permettono di accedere a tabelle e colonne utilizzando metodi specifici, generati automaticamente da Visual Studio.

Per esempio utilizzando un *DataSet* tipizzato si potrà scrivere:

```
MyDataSet.Clienti.RagioneSociale
```

invece di:

```
MyDataSet.Tables("clienti").Columns("RagioneSociale")
```

Un *DataSet* tipizzato consente una maggiore tranquillità in fase di codifica, in quanto permette di evitare errori nell'utilizzo dei nomi e dei tipi dei campi, che nel caso di un normale *DataSet* non possono essere intercettati dal compilatore.

Lo svantaggio, in questo caso, è dato da una eccessiva proliferazione di classi, e soprattutto dalla necessità di mantenerle a seguito di eventuali modifiche alla struttura della base dati.

In questo contesto, vedremo come sfruttare un *DataSet* tipizzato per implementare la logica applicativa, sfruttando il generatore di codice di Visual Studio e minimizzando quindi il numero di righe di codice da scrivere manualmente.

Per creare un *DataSet* tipizzato con Visual Studio si può procedere in questo modo:

- Anzitutto, si trascina su una form un oggetto *Data Adapter*. Nel nostro esempio viene utilizzato un piccolo database Access, e per questo motivo è stato scelto un *OleDbDataAdapter*. Le stesse operazioni, però possono essere compiute con gli altri *Data Adapter* disponibili.
- Utilizzando il *wizard* che viene avviato automaticamente, impostare le caratteristiche del *Data Adapter*, andando in particolare a specificare le tabelle di database da cui estrarre i dati. Nell'esempio allegato, viene utilizzata una semplice tabella *Automobile*.
- Infine, utilizzando il menu contestuale associato al nuovo *Data Adapter*, si seleziona il comando *Generate DataSet*, che consente di specificare il nome del *DataSet* tipizzato, nel nostro caso *AutomobileSet*, e lo si allega al progetto.

Questa operazione si traduce visivamente nella creazione di un file *AutomobileSet.xsd*, ovvero di uno schema XML che mantiene la struttura del *DataSet* corrispondente.

Oltre a questo file, però, viene creato anche un file "nascosto" *AutomobileSet.vb*, che può essere ispezionato passando alla visualizzazione di tipo *Class View*. Questo file contiene quattro classi. La principale è *AutomobileSet*, derivata da *DataSet*.

Le altre sono *AutomobileDataTable*, *AutomobileRow* e *AutomobileRowChangeEvent*, deri-

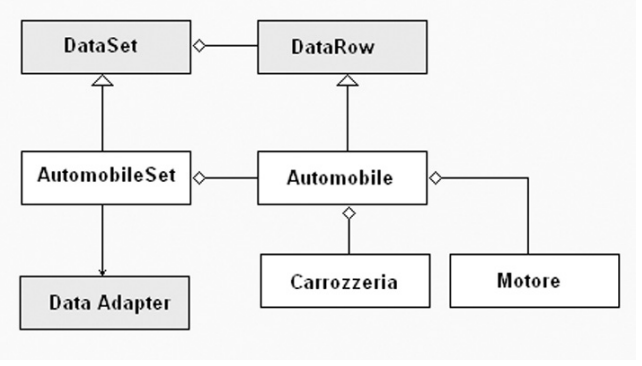


Figura 1 Architettura delle classi necessarie per implementare la logica applicativa

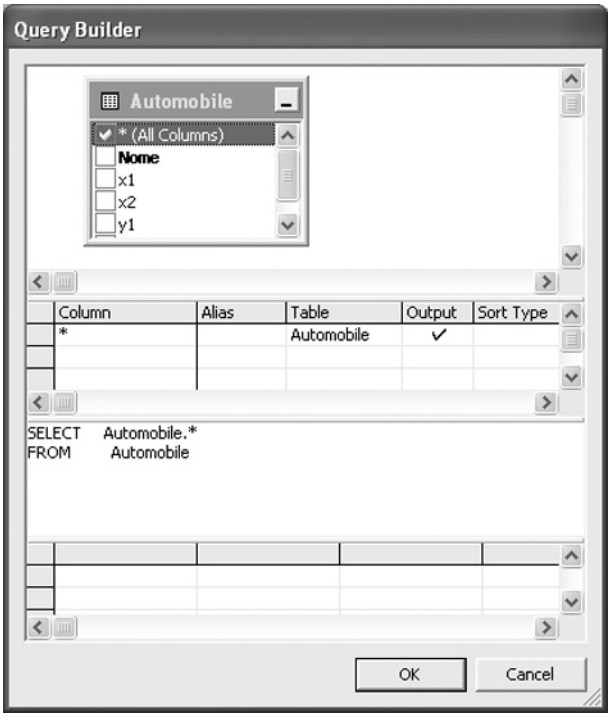


Figura 2 Il Query Builder permette di specificare interattivamente le caratteristiche di un Data Adapter

vate rispettivamente da *DataTable*, *DataRow* e *EventArgs*, e implementate come classi innestate di *AutomobileSet*.

Il codice di queste classi, generato automaticamente dal tool, è strutturalmente piuttosto semplice. La classe *AutomobileSet* contiene due costruttori specifici, e sovrascrive *(overrides*) alcuni metodi della classe base.

In più, contiene una proprietà *Automobile* che permette di accedere all'oggetto *DataTable* tipizzato.

La classe *AutomobileDataTable*, a sua volta, mette a disposizione un insieme di proprietà che permettono di accedere per nome alle varie colonne contenute nella tabella, nonché alcuni metodi, come *AddAutomobileRow*, *NewAutomobileRow* e *RemoveAutomobileRow*, che consentono di accedere alle righe della tabella considerandole come istanze della classe *AutomobileRow*.

Quest'ultima classe, infine, fornisce un insieme di proprietà tipizzate per accedere ai valori mantenuti nelle singole celle.

Gli oggetti applicativi

La classe *AutomobileRow* rappresenta in questo esempio l'elemento che ha il compito di mantenere la logica applicativa. Ogni sua istanza rappresenta il progetto di un'automobile, al quale devono poter essere applicate operazioni grafiche complesse, come *Capovolgi* e *Ruota*.

A titolo di esempio, il codice allegato implementa un semplice metodo *Sposta*, che permette di traslare l'oggetto a sinistra o a destra di un numero specificato di unità :

```
Public Sub Sposta(ByVal x As Integer)
    Me.x1 = Me.x1 + x
    Me.x2 = Me.x2 + x
End Sub
```

Grazie al fatto che questa classe è derivata da *DataRow*, sarà possibile utilizzare contemporaneamente sia i metodi che ne implementano la logica applicativa specifica, esemplificati da *Sposta*, sia quelli che consentono di sfruttare l'oggetto nella sua accezione generica di riga di una tabella.

> La scelta di sfruttare questo automatismo ci impone di accettare alcune limitazioni rispetto al progetto originale

L'applicazione di esempio

Il codice allegato contiene una piccola applicazione di esempio, costituita da un programma Windows che sfrutta le classi applicative *AutomobileSet* e *AutomobileRow*, mostrando come sia possibile accedere sia ai metodi specifici, sia a quelli generici.

In particolare, i dati mantenuti all'interno dell'oggetto *AutomobileSet* vengono normalmente

visualizzati all'interno di un *DataGrid*, mentre il codice associato ad un pulsante mostra come sfruttare i metodi applicativi specifici:

```
Dim oAuto As AutomobileSet.AutomobileRow
oAuto = AutomobileSet1.Automobile.Item(DataGrid1
                                       .CurrentRowIndex)
oAuto.Sposta(TextBox1.Text)
DataGrid1.Refresh()
```

L'accesso all'oggetto *oAuto* viene implementato tramite la proprietà *Item* della classe *AutomobileTable*, che restituisce un riferimento ad un oggetto *AutomobileRow*. Una volta ottenuto il riferimento a questo oggetto, diviene possibile richiamarne i metodi applicativi specifici.

La classe AutomobileRow rappresenta l'elemento che ha il compito di mantenere la logica applicativa

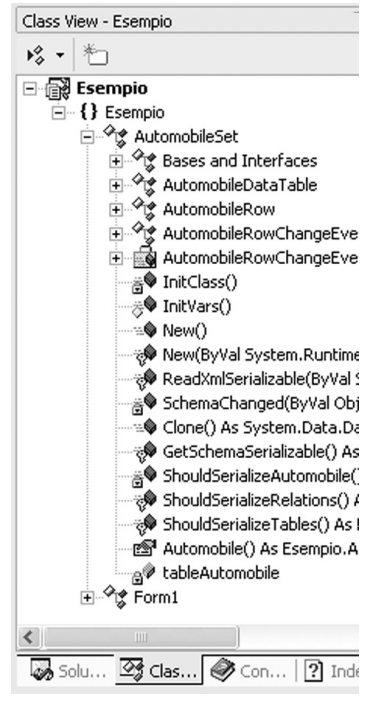


Figura 3 La vista Class View permette di accedere al file AutomobileSet.vb generato dal wizard

Conclusioni

Grazie all'automatismo offerto da Visual Studio nella generazione del codice associato ai *DataSet* tipizzati, l'implementazione di classi che incapsulino la logica applicativa del programma risulta semplice e veloce.

La scelta di sfruttare questo automatismo però ci impone di accettare alcune limitazioni rispetto al progetto originale.

Un *DataSet* tipizzato non è altro che una sottoclasse di *DataSet* con dei membri pubblici aggiuntivi

Anzitutto, il *Data Adapter* non viene incapsulato all'interno della classe, ma rimane al suo esterno. Inoltre, le classi generate vengono riempite di metodi non strettamente necessari ai nostri scopi, che, anche se molto utili per evitare errori a runtime, provocano la necessità di modificare il codice generato ogni qualvolta viene effettuata una modifica alla struttura del database. Queste limitazioni risultano però tutto sommato non troppo pesanti, soprattutto se si considera il risparmio di tempo che l'automatismo permette di ottenere. La soluzione che abbiamo ottenuto è molto .NET-centrica, e difficilmente portabile su diverse librerie, linguaggi di programmazione o ambienti di sviluppo.

Questa è stata però una scelta di progetto, che ha permesso di ottenere classi che, oltre a implementare una specifica logica applicativa, ci consentono di continuare a sfruttare gli automatismi del framework, come ad esempio la possibilità di mostrare i dati all'interno di una griglia.

Bibliografia

[1] L.Vandoni, "Implementazione della logica applicativa in programmi VB.NET", VBJ n. 62, Marzo 2005

[2] L.Vandoni, "L'accesso ai dati nei linguaggi object-oriented", VBJ n.60, Novembre 2004

Yukon in scena: si alza il sipario

Yukon, nome in codice della nuova release del motore di database di casa Microsoft, ovvero SQL Server 2005, sta prendendo definitivamente forma.

di Andrea Benedetti

In attesa della Beta 3, versione che ci si aspetta "features complete" e che dovrebbe vedere la luce nel prossimo mese di maggio, cominciamo a fare una panoramica su ciò che si presenterà agli occhi dei DBA.

L'avvento del Framework .NET ha rivoluzionato il modo di lavorare degli sviluppatori e molto probabilmente lo rivoluzionerà anche ai Database Administrator, grazie alla prossima versione del motore di database di casa Microsoft, completamente integrato con la prossima release del Common Language Runtime (la 2.0). SQL Server 2005 introduce, finalmente, molte funzionalità attese e richieste dalla comunità degli sviluppatori/utilizzatori, alcune delle quali già presenti su sistemi concorrenti.

La scelta di versione su cui lavorare per la stesura dell'articolo è caduta sulla CTP (Community Technology Preview) di Dicembre 2004, la versione 9.00.0981, l'ultima uscita dai laboratori di sviluppo Microsoft (**Figura 1**). Applicazione sicuramente meno stabile della Beta2 già uscita, ma versione che introduce nuove funzionalità (una tra tutte quella di aprire e visualizzare le tabelle - Open Table, come nel vecchio Entreprise Manager - anche se ancora non consente la distinzione tra "Return all rows" e "Return Top").

I nuovi tool di amministrazione e gestione

Due sono le novità relative ai tool di gestione e di amministrazione presenti in questa nuova versione: *SQL Computer Manager*, uno snap-in per la Microsoft Management Console (MMC), e *SQL Server Management Studio*.

Andrea Benedetti *si occupa di disegno e manutenzione di basi di dati e di sviluppo di applicazioni web–based finalizzate all'erogazione di servizi sanitari per conto di Studio–farma Srl di Brescia Può essere contatto all'indirizzo e–mail: abenedetti@infomedia.it*

Il primo, *SQL Computer Manager*, è un unico strumento che raccoglie al suo interno le varie utilità di servizi e di protocolli prima distribuiti in diversi software (**Figura 2**).

Dalla console è possibile definire alias di client (il vecchio Client Network Utility, ora chiamato SQL Native Client Configuration), protocolli di comunicazione abilitati e gestire tutti i servizi che possiamo installare a seconda della versione del software: naturalmente SQL Server e SQL Agent e poi Analysis Server, Full Text, Report Server. Da questa console interagiamo direttamente con tutti i servizi sopra elencati, modificandone la modalità di esecuzione (manuale, automatica o disabilitata), la password di accesso e le classiche operazioni di start – stop – restart.

Per rispondere ad una domanda che ricorre spesso sui newsgroup, è utile specificare che tale utilità è legata alla macchina, quindi al server locale, che sia mono o multi istanza. Non è quindi possibile effettuare registrazioni verso altri server. La seconda, più importante e chiaramente più evidente novità, è *SQL Server Management Studio*.

Un unico software costruito ad hoc per la gestione e l'amministrazione dei server, che unisce

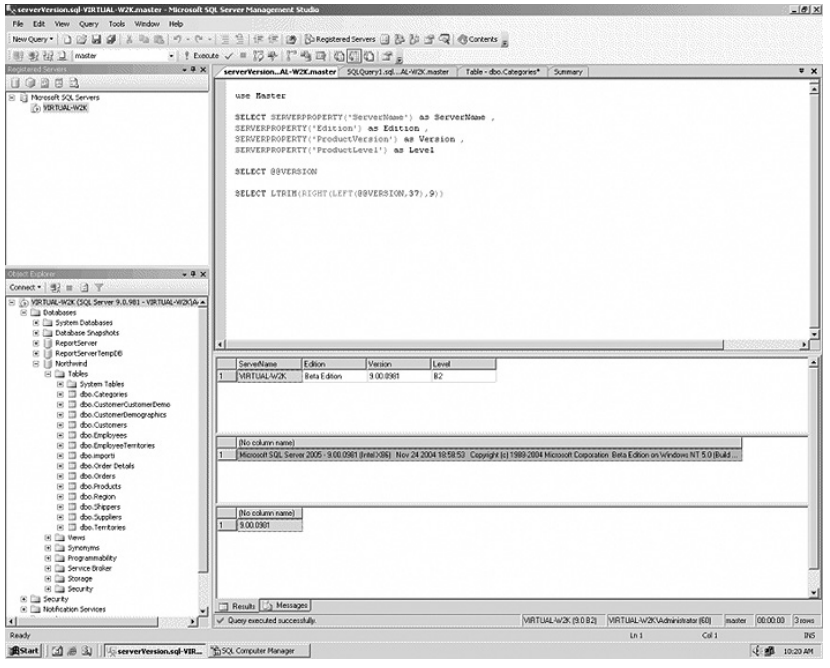


Figura 1 SQL Server Management Studio

la nostra istanza, scegliamo la voce "Attach".

Si presenterà a video una finestra che, pur essendo diversa dalla versione 2000, ci consente di capire immediatamente che dobbiamo premere il pulsante "Add" per selezionare il file di dati del database di cui vogliamo eseguire l'attach e premere "Ok". Selezioniamo quindi il file northwnd.mdf e procediamo al suo attach. Tre sono le novità che spiccano subito in questa finestra:

- la finestra di selezione file adesso non è più una form-modale;
- consente di visualizzare sotto forma di script l'azione che stiamo effettuando;
- consente la possibilità di schedulare (e quindi di salvare) l'azione per eseguirla in un secondo momento.

Si tratta di novità che non sono presenti solo nella finestra di selezione file, ma che possiamo ritrovare praticamente in tutte le attività che siamo in grado di svolgere sul server. La possibilità di avere form non - modali (con il loro pulsante sulla barra delle applicazioni) ci consente, ad esempio visualizzando le proprietà di una tabella, di effettuare delle comparazioni a video tra oggetti di database/server diversi (cosa impossibile con la finestra delle proprietà nella versione precedente). La possibilità di salvare (potendo scegliere tra file, memoria, o una nuova finestra di query) lo script delle azioni che andremo a compiere, consente in primis di capire come si comporta il software dietro le quinte, quindi di poter sapere quali saranno le istruzioni che verranno effettivamente eseguite.

le funzionalità che erano proprie sia dell'Enterprise Manager sia del Query Analyzer che quindi, in quanto software distinto e separato, non esiste più. Abbandonata la scelta di utilizzare uno snap-in da inserire nella Microsoft Management Console (com'era invece Enterprise Manager) il tool è stato completamente riscritto e ridisegnato, con un'interfaccia utente che, chi è già abituato all'uso di Visual Studio, troverà assai familiare.

È interessante sapere da subito che il motore di SQL Server 2005 consente la gestione anche di server relativi alla precedente versione (la 2000) e non viceversa, così come consente l'attach di file di database provenienti proprio da quella versione.

E proprio per effettuare una panoramica delle funzionalità e cominciare così a prendere dimestichezza con il nuovo strumento, eseguiamo un attach dei file del classico database Northwind prelevato da un'istanza di SQL Server 2000.

Come avremmo fatto con la versione precedente, e quindi in maniera assolutamente immediata ed intuitiva, cliccando con il tasto destro del mouse sulla cartella Databases del-

Anche nella versione precedente era possibile fare questo, a patto di aprire una traccia con SQL Profiler ed analizzare i vari comandi eseguiti passo passo. Avendo parlato dell'attach di un database, è utile aprire subito una parentesi sull'operazione inversa: il detach. Questa operazione, che similmente alla precedente può essere fatta con il tasto destro sul database interessato, menu Tasks – Detach, consente di poter chiudere (drop) le connessioni presenti ed eventualmente di aggiornare le statistiche della base dati. Una volta installato il nostro database, prima di espandere il nodo "Northwind" appena creato, clicchiamo su di esso per visualizzare un'altra novità: la vista Summary (tasto F7). Una visualizzazione che risulta familiare a chi era già abituato al "TaskPad" della versione precedente. Vista, questa, che ci permette di avere sotto controllo informazioni circa lo spazio allocato, utilizzato e rimanente, data dell'ultimo backup, connessioni attive e data di creazione. Espandiamo adesso il nodo "Northwind" per accorgerci della nuova suddivisione, quindi dell'organizzazione logica, che viene fatta dei vari oggetti. Nei vari nodi figlio, se restano subito visibili sia Tables che Views, possiamo notare una nuova cartella – Programmability – che contiene al suo interno le stored procedure, le funzioni, i trigger (finalmente si possono vedere tutti e subito, senza dover cercare tramite *tasto destro – all tasks – manage triggers* sulle tabelle), tipi e ruoli, valori di default, assembly. Da sottolineare come le varie cartelle di oggetti contengano già, per ogni database, le stored procedure, le funzioni e le viste di sistema, consentendo, con un semplice drag & drop sulle finestre di query, di inserirle nelle istruzioni che costruiremo (ricorda, anche se in maniera molto meglio organizzata, l'Object Browser del vecchio Query Analyzer). Un'altra funzione accessibile direttamente dal menu contestuale all'oggetto database (tasto destro sul db, Tasks, Copy Database) è la copia di database.

La scelta di questa funzione scatena la partenza di un wizard, quindi di un percorso guidato, che consente la copia di un intero database da una istanza SQL 2000 o naturalmente SQL 2005, ad una istanza SQL 2005.

La copia può essere fatta adesso scegliendo tra due differenti modalità:

- utilizzando il metodo detach/attach
- utilizzando il metodo di trasferimento SQL Management Object

Il secondo metodo, certamente più lento, è chiaramente da preferirsi in tutti quei casi in cui si è impossibilitati a mettere in stato di offline il database di partenza.

Il metodo si rivela perfetto anche per effettuare copie di database sulla stessa istanza, semplicemente cambiando il nome del db e dei file (o il percorso) di destinazione. Utilissimo il nuovo Template Explorer (dal menu View, oppure: CTRL + Alt + T) dal quale è possibile recuperare modelli guidati di istruzioni da eseguire: attach/detach di database, configurazione procedure di invio mail, definizione trigger, ecc. (**Figura 3**).

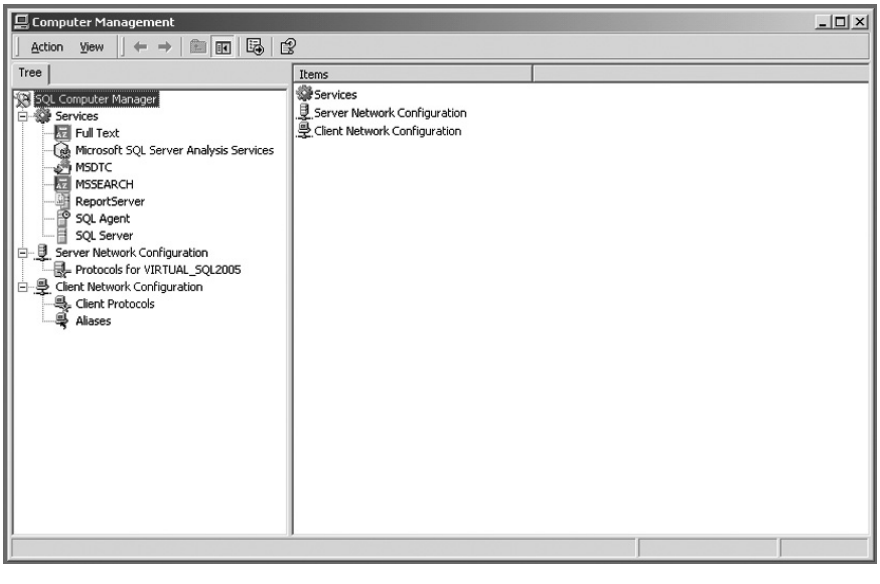


Figura 2 SQL Computer Manager

Gli attori entrano in scena

Certamente le novità introdotte non riguardano solo gli strumenti di amministrazione e gestione. Facciamo allora una panoramica cercando di presentare alcune delle più interessanti. La prima, sicuramente quella più attesa e pubblicizzata, riguarda, come abbiamo già accennato all'inizio, il supporto e l'integrazione con il framework .NET (la prossima release di Visual Studio 2005 contiene un nuovo tipo di progetto "Yukon"), supporto dato non certo per mandare in pensione il linguaggio T-SQL, che resta il linguaggio principe per la scrittura, il recupero e la manipolazione dei dati, quanto per poter ottenere benefici che si potranno avere, ad esempio, con l'accesso diretto a funzioni di crittografia, interazione con oggetti COM tramite wrapper, possibilità di scrivere ed utilizzare funzioni che prima avremmo dovuto scrivere ed installare come stored procedure estese, accedere a risorse esterne (file, Web Service…) e, comunque, possibilità di utilizzare tutto ciò che il framework mette a disposizione. Attenzione: il CLR (Common Language Runtime) è disabilitato per default e quindi per poterlo utilizzare è necessario abilitarlo, ad esempio tramite la procedura *sp_configure*, come:

```
EXEC sp_configure 'show advanced options' , '1'
go
reconfigur
go
EXEC sp_configure 'clr enabled' , '1'
```

Quindi possiamo costruire oggetti che si riferiscono a metodi di classe, come:

```
CREATE Function / Procedure / Trigger AS
EXTERNAL NAME myAssembly.myClass.myMethod
```

Le utilità oSql e iSql vengono sostituite e potenziate da un unico nuovo tool sempre a riga di comando: SQLCMD

```
C:\>SQLCMD -E -dNorthwind -q"select * from region"
```

Il supporto mail è stato molto migliorato e, seppur esistendo ancora SQL Mail, entra in scena SQLiMail, una nuova gestione non più basata su MAPI ma su protocollo e server SMTP (consentendo di specificarne anche più di uno). Il modello ad oggetti DMO, pur restando per mantenere la compatibilità verso il basso, viene rimpiazzato ed ampliato da due nuovi modelli ad oggetti, completamente scritti per .NET, SMO (SQL Server Management Objects)e RMO (Replication Management Objects), ovvero il modello relativo alla gestione delle repliche che saranno, da adesso, possibili anche su protocollo HTTP L'utility Index Tuning Wizard viene rimpiazzata ed ampliata dal DTA, Database Tuning Advisor, che analizzando stress ed utilizzo del server, fornisce raccomandazioni circa l'aggiunta, la modifica e la rimozione di strutture legate alle performance (indici cluster e non-cluster, viste indicizzate, ecc.). Compare la

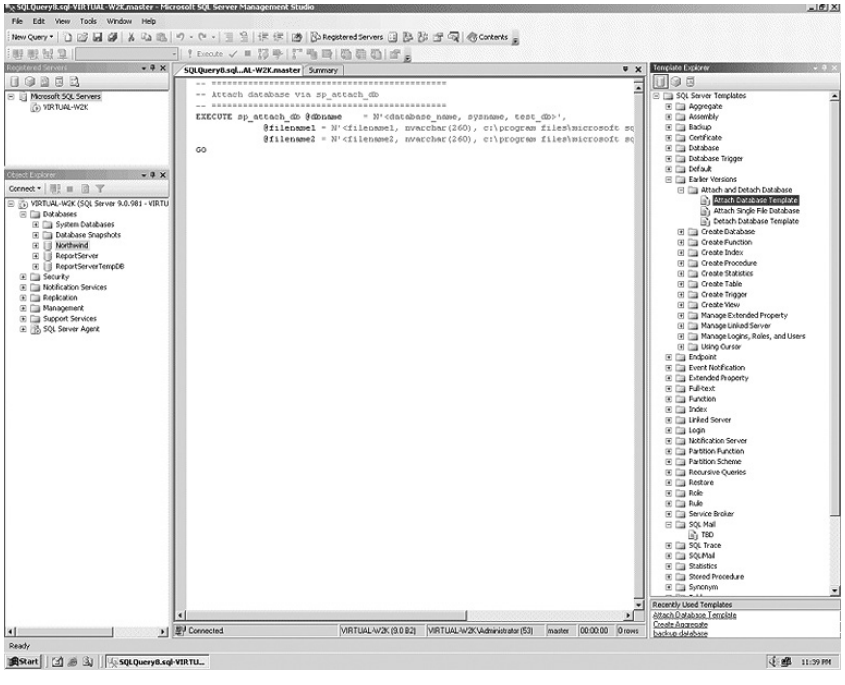


Figura 3 Il Template Explorer

funzione ROW_NUMBER, in grado di restituire un numero sequenziale di righe, comodissima per risolvere il classico problema della paginazione dei dati su applicazioni web-based:

```
USE Northwind
GO
WITH myProducts AS
(
SELECT ProductName, UnitPrice,
ROW_NUMBER() OVER (order by ProductID)as RowNumber
FROM products
)
SELECT * FROM myProducts
WHERE RowNumber between 40 and 50
```

Nasce il nuovo operatore PIVOT, che consente di realizzare molto facilmente query incrociate (possibili anche con la versione precedente, a patto di lavorare molto sulla scrittura di procedure altamente personalizzate).

Attenzione: per poter utilizzare questo operatore su database provenienti dalla versione 2000 di SQL Server, è necessario che la compatibilità del database sia impostata a 90 (ovvero sul modello SQL Server 2005). Per vedere la compatibilità impostata è sufficiente eseguire la procedura *sp_dbcmptlevel*, come:

```
EXEC sp_dbcmptlevel @dbName ='Northwind'
```

Quindi, se utilizziamo per la nostra prova il database Northwind importato da una versione precedente, dobbiamo portare il suo livello di compatibilità a 90 (**Figura 4**):

```
USE [master]
GO
EXEC dbo.sp_dbcmptlevel @dbname=N'Northwind',
                                    @new_cmptlevel=90
GO
```

e poi provare l'operatore:

```
USE Northwind

SELECT EmployeeID, [1996] as Y1, [1997] as Y2
from
(select 1 as num, year(orderDate) as Anno, EmployeeID
                                        from orders) o
PIVOT
(count (num) for Anno IN ([1996],[1997])
) as pvt
order by EmployeeID
```

Il tipo di dati varchar che nella versione 2000 permetteva una memorizzazione massima di 8.000 caratteri consente, specificando la clausola *max*, di ampliarne la dimensione memorizzabile fino ad un massimo di 2 GB introducendo, di fatto, il concetto di tipi "Large Value", arrivando alla memorizzazione di (2^{31} – 1) byte.

```
CREATE TABLE tabTest
( id int primary key, nome varchar(35), note varchar(max) )
```

La stessa clausola *max* può essere applicata anche ai tipi Varbinary.

Questa novità fornisce un'alternativa all'utilizzo dei tipi text, ntext ed image, uniformando il modello di accesso ai dati con i tipi già presenti (possiamo quindi evitare di ricorrere a funzioni specifiche costruite per i tipi blob: writeText, readText, updateText). Vengono introdotti trigger che possono essere scatenati su eventi legati al database (alter/create/drop database), in sostanza su istruzioni di Data Definition Language (DDL).

Questa tipologia di trigger, pur venendo invocata sempre dopo lo scatenarsi dell'istruzione (ovvero non esistono, per questa tipologia, trigger di tipo instead of), può comunque essere utilizzata per prevenire modifiche o cancellazioni di tabelle, come nell'esempio seguente:

```
USE dbTest
go
-- Creazione trigger su drop table
CREATE TRIGGER trDeleteTable
ON DATABASE
FOR DROP_table
AS
  PRINT 'Impossibile cancellare la tabella!'
  ROLLBACK
GO
-- Se proviamo la drop table:
DROP TABLE tabTest

/*
Questo il messaggio che riceviamo:

Impossibile cancellare la tabella!
Msg 3609, Level 16, State 2, Line 1
Transaction ended in trigger. Batch has been aborted.
*/
```

Gestione degli errori, come per i linguaggi del Framework .Net, a blocchi try … catch

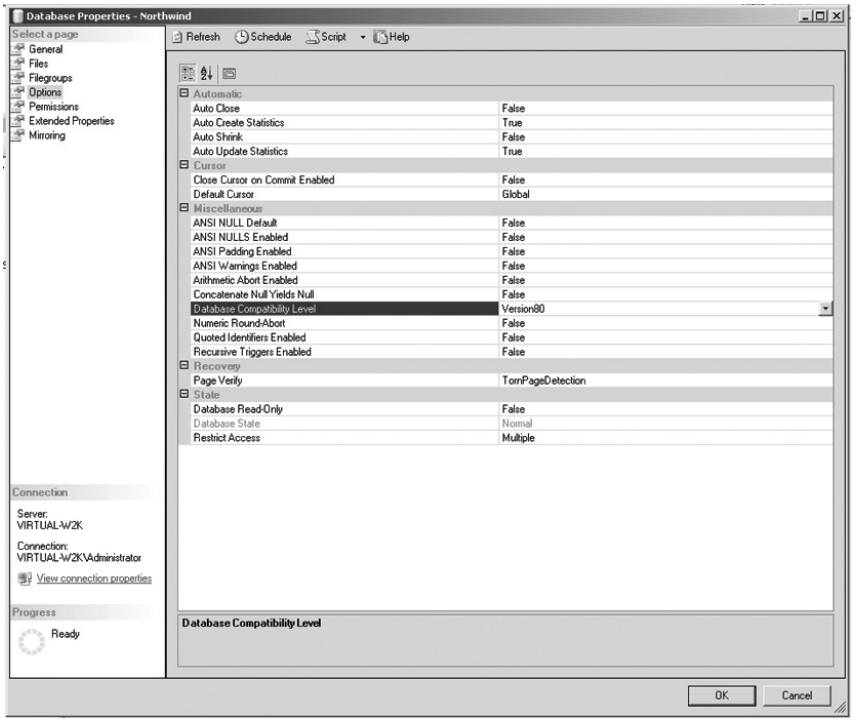


Figura 4 Proprietà e opzioni del database

```
BEGIN TRANSACTION;
GO

BEGIN TRY
  -- Generiamo un errore:
  SELECT 4/0
   print 'tutto ok'
END TRY
BEGIN CATCH
  print 'errore'
END CATCH
GO

ROLLBACK TRANSACTION;
GO
```

Nell'esempio appena riportato, all'interno del blocco try, si genera un'eccezione che viene correttamente gestita dal blocco catch.

Viene riscritto il modello di trasformazione dati (ex DTS) in SQL Server Integration Services. Chiudiamo la presentazione con altre tre importanti novità legate ai database ed ai loro dati:

- *Database mirroring* (disponibile solo con database con recovery model impostato a full): consente di mantenere aggiornato un secondo server. In caso di caduta del server di produzione, le applicazioni sono in grado di riconnettersi al secondo server.
- *Database snapshot* (disponibile solo nella versione enterprise): consente di effettuare un'istantanea, in sola lettura, dei dati, ideale per essere utilizzata in soluzioni di reportistica.
- *Data partitioning*: consente di effettuare split di tabelle di grosse dimensioni, aumentando performance e semplificando le operazioni di manutenzione.

Conclusioni

Le novità e le migliorie non finiscono certamente con l'elenco e la panoramica esposta nell'articolo. SQL Server 2005 aggiunge alla versione precedente molte funzionalità in grado di rendere migliore la vita a sviluppatori ed amministratori di database: programmabilità, reportistica, soluzioni di fault tolerance e disaster recovery, supporto ai device mobili, XML, XML schema, XQuery, Web Service, Notification Services …

Intanto abbiamo cominciato a presentare ed a conoscere alcuni dei principali attori tra quelli che entreranno in scena. Ma lo spettacolo, signori, deve ancora cominciare. E se il trailer già vi tenta, la pellicola sarà ancora più interessante.

Riferimenti

[1] http://www.microsoft.com/sql/2005/default.asp
[2] newsgroup: microsoft.public.it.sql

Log delle transazioni mediante trigger

In particolari contesti, è utile che l'applicazione software garantisca una risposta a domande del tipo: "Chi ha modificato il tale attributo della tale tabella, e quando ciò è avvenuto?". Un log delle transazioni garantisce una risposta adeguata a tali esigenze.

di Francesco Quaratino

Un database OLTP (On-Line Transaction Processing), per sua natura, registra le transazioni non conservando memoria alcuna dei cambiamenti avvenuti nel tempo.

> In determinate situazioni, assume un'importanza particolare la possibilità di risalire al momento esatto e all'utente che ha fatto scaturire una cancellazione piuttosto che un aggiornamento o un inserimento

Non è possibile quindi tracciare la storia di un'entità del database, da cui emergano informazioni circa i mutamenti dei suoi dati nel tempo e ad opera di chi essi siano avvenuti. Intendo riferirmi, quindi, ad una vera e propria operazione di *logging* attraverso cui risalire, mediante semplici interrogazioni, allo stato di un'entità di database, incluso l'utente di database che lo ha determinato e le sue coordinate temporali. È possibile imbattersi in diverse situazioni in cui assume un'importanza particolare, e talvolta cruciale, la possibilità di risalire al momento esatto e all'individuo, o più genericamente all'utente, che ha fatto scaturire una cancellazione piuttosto che un aggiornamento o un inserimento. Un tale sistema di log può essere implementato sia a livello di applicazione che a livello di database.
Questo articolo propone una risposta a questa esigenza, con una soluzione a livello di database implementata in SQL Server 2000 mediante l'uso di trigger.

Francesco Quaratino *è consulente in ambito di progettazione e amministrazione di database OLTP e OLAP, oltre che di applicazioni software orientati ai dati. È certificato MCDBA e sviluppa in VB6, ASP, VB.NET e ASP.NET. Può essere contatto all'indirizzo e–mail: fquaratino@infomedia.it*

Fattori di rischio

Progettare una soluzione di log che registri le transazioni in un database piuttosto che in un formato testo o XML, pone il progettista del sistema di fronte ad alcuni fattori la cui analisi preventiva può risultare fondamentale per la sua buona riuscita. Registrare ogni cambiamento avvenuto nell'arco di vita di una o più entità di un database operazionale, implica il far fronte a costi di manutenzione delle procedure software realizzate a tale scopo - sia a livello di applicazione che a livello di database -, come anche a maggiori costi di manutenzione dei database coinvolti e di ottimizzazione delle performance - si pensi, per esempio, alla memoria su disco necessaria ad ospitare i dati di log e al fatto che ad ogni operazione di inserimento/modifica/cancellazione nel database operazionale ne

corrisponde una nel database di log.

Sarà necessario valutare l'impatto della procedura di log sulle performance del database operazionale, ed eventualmente considerare una procedura di storicizzazione o di aggregazione dei dati di log i quali potrebbero raggiungere dimensioni su disco considerevoli. Una soluzione che prevede un database come *repository* dei dati di log - come quella proposta in questo articolo -, fa nascere problematiche simili a quelle che scaturiscono nella progettazione dei più comuni database OLTP, perché sarà soggetto a frequenti scritture. Mi riferisco, per esempio, alla scelta dei campi da indicizzare, che in un database OLTP privilegia i dati il cui tipo occupa quanto meno spazio possibile, allo scopo di rendere più efficienti le operazioni di scrittura; questo perché in un sistema di registrazione delle transazioni saranno di gran lunga più numerose le operazioni di scrittura rispetto a quelle di lettura. Per lo stesso motivo, la creazione di un qualsiasi indice sarà il risultato di una scelta oculata e più che giustificata, e nella manutenzione degli indici si imposterà un FILLFACTOR (fattore di riempimento) ragionevolmente basso. A questo proposito, ricordo che tanto più il fillfactor di un indice - sia esso clustered o meno - è prossimo al 100% tanto meno spazio libero viene riservato nelle pagine di dati dell'indice per i futuri inserimenti. Un valore alto di fillfactor causa frequenti *split* dei dati dell'indice che incidono negativamente sui tempi di esecuzione di operazioni di scrittura. Per impostare il valore di riempimento basta eseguire il comando di ricostruzione degli indici (DBCC DBREINDEX) specificando il nome del database e opzionalmente il nome dell'indice specifico, per esempio:

```
DBCC DBREINDEX ('MyDB', my_index_name, 65)
```

È importante notare che il valore del fillfactor, così come lo stato di frammentazione di un indice, si altera col tempo e si ristabilisce unicamente con la ricostruzione dell'indice stesso.

Per approfondire queste ad altre considerazioni circa la progettazione degli indici di SQL Server 2000, rimando all'ottimo eBook disponibile gratuitamente per il download dal sito [1].

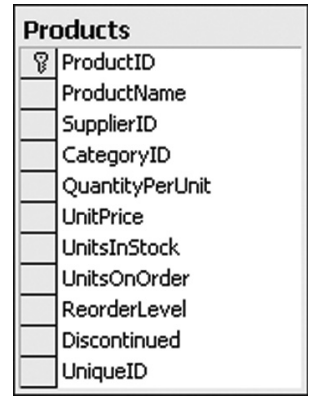


Figura 1 Tabella del database di esempio

Listato 1 Script di creazione delle tabelle per ospitare il log di [Products]

```
USE NorthwindLog
GO

CREATE TABLE [dbo].[Products_InsertedDeletedLog] (
   [PKey] [int] IDENTITY (1, 1) PRIMARY KEY CLUSTERED,
   [ProductID] [int] NOT NULL,
   [InsertUser] [nvarchar] (256) NOT NULL,
   [InsertTime] [datetime] NOT NULL,
   [DeleteUser] [nvarchar] (256) NULL,
   [DeleteTime] [datetime] NULL,
   [UniqueID] [uniqueidentifier] NOT NULL )
GO

CREATE TABLE [dbo].[Products_UpdatedLog] (
   [PKey] [int] IDENTITY (1, 1) PRIMARY KEY CLUSTERED,
   [FKey] [int] NOT NULL,
   [ProductID] [int] NOT NULL,
   [ProductName] [nvarchar] (40),
   [SupplierID] [int],
   [CategoryID] [int],
   [QuantityPerUnit] [nvarchar] (20),
   [UnitPrice] [money],
   [UnitsInStock] [smallint],
   [UnitsOnOrder] [smallint],
   [ReorderLevel] [smallint],
   [Discontinued] [bit],
   [UpdateUser] [nvarchar] (256) NOT NULL,
   [UpdateTime] [datetime] NOT NULL )
GO

CREATE  INDEX [IX_ProductID]
   ON [dbo].[Products_InsertedDeletedLog]([ProductID])
GO

CREATE  INDEX [FK_FKey]
   ON [dbo].[Products_UpdatedLog]([FKey])
GO
```

Un'architettura possibile

Supponiamo di voler tracciare il log della tabella [Products] del database di esempio [Northwind] (**Figura 1**) e di volerlo fare in un database diverso

da quello operazionale - quest'ultima scelta potrebbe avere diverse motivazioni che sono fuori dallo scopo di questo articolo, in cui intendo, comunque, dimostrare la sua fattibilità. Innanzitutto, creiamo un nuovo database [NorthwindLog] e disegniamo le tabelle che ospiteranno il log della tabella [Products] (**Listato 1**): la [Products_InsertedDeletedLog] che registrerà la nascita e morte del prodotto (eventi che coincidono con l'inserimento e la cancellazione) e la [Products_UpdatedLog] che registrerà invece i cambiamenti avvenuti durante la vita del prodotto (che coincidono con l'inserimento e gli aggiornamenti) (**Figura 2**). Analizziamo in dettaglio la struttura delle due tabelle e la relazione che le lega. [Products_InsertedDeletedLog] presenta la chiave primaria surrogata [PKey] - comodamente autogestita attraverso l'autoincremento del tipo di dato IDENTITY -, il [ProductID] che rappresenta la chiave primaria della tabella sorgente [Products], i campi [InsertedUser] e [InsertedDate], in cui saranno registrati rispettivamente l'utente di database e la data e ora dell'inserimento del prodotto, e [DeletedUser] e [DeletedDate], rispettivamente l'utente di database e la data e ora della cancellazione del prodotto.

> I trigger sono utilizzati per mantenere integri i dati in database normalizzati e non, o per implementare logiche applicative molto complesse

Vi è un ultimo campo [UniqueID] di tipo UNIQUEIDENTIFIER che sarà discusso opportunamente nel seguito dell'articolo.

La chiave primaria surrogata [PKey] ha la funzione di identificare univocamente il prodotto, superando il limite rappresentato dalla possibile modifica del valore della chiave primaria della tabella di cui si intende tracciare il log.

Questo non è il caso della tabella [Products] del nostro esempio, la cui chiave primaria è di tipo IDENTITY, ma supponendo che [ProductName] fosse stata la chiave primaria e che il suo valore fosse stato modificato per uno o più prodotti, il nostro log avrebbe dovuto tenerne traccia considerandolo però non come un nuovo prodotto, bensì un prodotto già in precedenza creato e a cui si fossero apportate modifiche. La tabella [Products_UpdatedLog] presenta tutti i campi della tabella [Products] con l'aggiunta di una chiave primaria [PKey] di tipo IDENTITY, una chiave esterna [FKey] alla chiave primaria [PKey] di [Products_InsertedDeletedLog], i campi [UpdateUser] e [UpdateTime] in cui

Listato 2 Il trigger per l'inserimento del log in fase di creazione N prodotti

```
USE Northwind
GO

CREATE TRIGGER tr_insert_Products ON [dbo].[Products]
FOR INSERT
AS
DECLARE @date datetime
BEGIN

IF @@ROWCOUNT = 0
  RETURN

SET @date = GETDATE()

INSERT INTO NorthwindLog.dbo.Products_InsertedDeletedLog
  (ProductID, InsertUser, InsertTime, UniqueID)
SELECT ProductID, USER, @date, UniqueID
FROM Inserted

INSERT INTO NorthwindLog.dbo.[Products_UpdatedLog]
  (FKey, ProductID, ProductName, SupplierID, CategoryID,
  QuantityPerUnit, UnitPrice, UnitsInStock, UnitsOnOrder,
  ReorderLevel, Discontinued, UpdateUser, UpdateTime)
SELECT
  IDL.PKey, I.ProductID, I.ProductName, I.SupplierID, I.CategoryID,
  I.QuantityPerUnit, I.UnitPrice, I.UnitsInStock, I.UnitsOnOrder,
  I.ReorderLevel, I.Discontinued, USER, @date
FROM Inserted AS I INNER JOIN NorthwindLog.dbo.Products_InsertedDeletedLog AS IDL
ON I.UniqueID = IDL.UniqueID

END
GO
```

saranno registrati l'utente di database e la data e l'ora dell'inserimento/aggiornamento. Quindi, per la buona riuscita dell'operazione di logging, ecco quanto deve avvenire in sintesi: per ogni prodotto inserito in [Produtcs], si crea una riga in [Products_InsertedDeletedLog] che ne registra i dati di nascita (chiave primaria del prodotto, utente e data inserimento), con una chiave primaria surrogata; in [Products_UpdatedLog] si inserisce una riga che riporta lo stato di ogni singolo campo nell'operazione di inserimento; le eventuali successive modifiche a uno qualsiasi dei campi di [Products], producono l'inserimento di una riga in [Products_UpdatedLog] che conserva la relazione con la [Products_InsertedDeletedLog] attraverso la chiave esterna [FKey]; infine, la cancellazione di un prodotto in [Products] comporta l'aggiornamento dei campi [DeletedUser] e [DeletedDate] della riga di [Product_InsertedDeletedLog.

Listato 3 Il trigger per l'inserimento del log in fase di aggiornamento di N prodotti

```
CREATE TRIGGER tr_update_Products ON [dbo].[Products]
FOR UPDATE
AS
DECLARE @date datetime
BEGIN

IF @@ROWCOUNT = 0
  RETURN

SET @date = GETDATE()

INSERT INTO NorthwindLog.dbo.[Products_UpdatedLog]
  (FKey, ProductID, ProductName, SupplierID, CategoryID,
  QuantityPerUnit, UnitPrice, UnitsInStock, UnitsOnOrder,
  ReorderLevel, Discontinued, UpdateUser, UpdateTime)
SELECT
  IDL.PKey, I.ProductID, I.ProductName, I.SupplierID, I.CategoryID,
  I.QuantityPerUnit, I.UnitPrice, I.UnitsInStock, I.UnitsOnOrder,
  I.ReorderLevel, I.Discontinued, USER, @date
FROM Inserted AS I INNER JOIN NorthwindLog.dbo.Products_InsertedDeletedLog AS IDL
ON I.UniqueID = IDL.UniqueID

END
GO
```

> La chiave primaria surrogata ha lo scopo di superare il limite derivante dalla possibile modifica del valore della chiave primaria della tabella di cui tracciare il log

Usare i trigger

Prima di analizzare i trigger creati per implementare il sistema di log, è opportuno analizzare un problema, alla cui soluzione si è solo accennato, circa la possibilità di effettuare un inserimento di massa, per esempio:

```
INSERT INTO [Products] (ProductName, QuantityPerUnit, UnitPrice)
SELECT ProductName, QuantityPerUnit, UnitPrice
FROM [Products2004]
```

In un trigger AFTER INSERT sulla tabella [Products], la pseudo-tabella Inserted è costituita da più righe risultanti dalla SELECT. In questo caso, non è possibile risalire al valore del campo [PKey] autogeneratosi con incremento automatico, poiché la pseudo-tabella Inserted è un insieme non ordinato di righe (come ogni altra tabella di un database relazionale). Per il nostro trigger, conoscere il valore assegnato ad ogni [PKey] inserito, è fondamentale al fine di stabilire una relazione tra [Products_InsertedDeletedLog] e [Products_UpdatedLog] attraverso i campi [PKey] e [FKey]. Il problema è stato risolto aggiungendo il campo [UniqueID] di tipo UNIQUEIDENTIFIER nella tabella di origine, con un vincolo di DEFAULT che ne imposta il valore attraverso la funzione NEWID(), il quale genera un identificatore univoco globale (GUID, Globally Unique Identifier):

```
ALTER TABLE [Northwind].[dbo].[Products]
   ADD UniqueID UNIQUEIDENTIFIER DEFAULT(NEWID())
GO
```

Più semplicemente si potrebbe usare un semplice tipo IDENTITY con autoincremento, ma la scelta è caduta sulla prima soluzione per rendere più chiara la distinzione tra il campo [UniqueID] e

[ProductID]. Il **Listato 2** mostra il trigger AFTER INSERT che alimenta il database di log durante la fase di creazione di un nuovo prodotto. In questo modo, per ogni nuovo prodotto si ha un identificatore univoco rappresentato dal campo [UniqueID] che consente di stabilire una relazione tra le due pseudo-tabelle Deleted e Inserted nel trigger AFTER UPDATE (**Listato 3**). In esso, si noti come la JOIN tra Deleted e [Products_InsertedDeletedLog] sia costruita sui due campi [UniqueID] delle rispettive tabelle. Per finire, il trigger AFTER DELETE (**Listato 4**) produce un aggiornamento dei campi [DeleteUser] e [DeleteTime] della riga di [Products_InsertedDeletedLog]. Una vista sul database consente di leggere i dati del log, ottenendo per ogni riga tutte le informazioni utili: data di inserimento, aggiornamento e cancellazione, oltre che lo stato di ogni singolo campo nell'ambito di una operazione di trattamento dei dati.

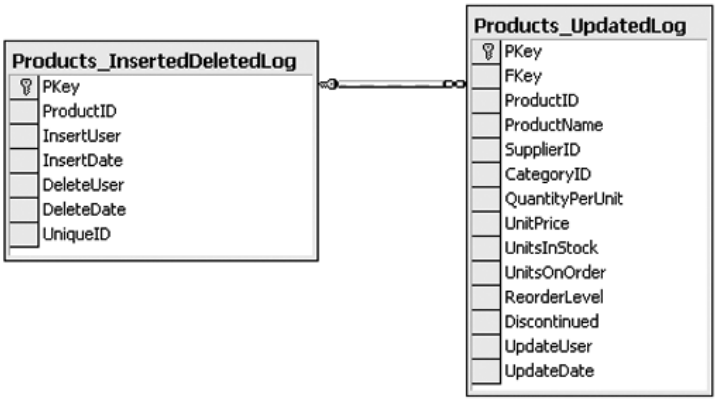


Figura 2 Le nuove tabelle create

```
CREATE VIEW dbo.[ProductsLog_VIEW]
AS
SELECT TOP 100 PERCENT
P_IDL.ProductID AS FirstProductID, P_UL.*,
InsertUser, InsertTime, DeleteUser, DeleteTime
FROM dbo.Products_InsertedDeletedLog P_IDL
INNER JOIN dbo.Products_UpdatedLog P_UL
ON P_IDL.PKey = P_UL.FKey
ORDER BY P_UL.FKey, P_UL.PKey
```

Listato 4 Il trigger per l'inserimento del log in fase di eliminazione di N prodotti

```
CREATE TRIGGER tr_delete_Products ON
[dbo].[Products]
FOR DELETE
AS
DECLARE @date datetime
BEGIN

IF @@ROWCOUNT = 0
  RETURN

SET @date = GETDATE()

UPDATE IDL
SET DeleteUser = USER, DeleteTime = GETDATE()
FROM Deleted As D INNER JOIN NorthwindLog.dbo.Products_
                             InsertedDeletedLog AS IDL
ON D.UniqueID = IDL.UniqueID

END
GO
```

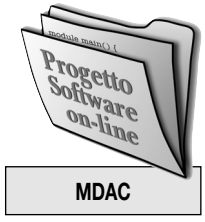
Conclusioni

L'idea di partenza era quella di avere in tempo reale informazioni sullo stato di un'entità del database. Per realizzarla si è pensato all'uso degli AFTER TRIGGER, i quali sono solitamente utilizzati per mantenere integri i dati in database normalizzati e non, o per implementare logiche applicative particolarmente complesse [2] [3]. Nel caso affrontato in questo articolo i trigger si sono dimostrati – ancora una volta - un approccio vincente alla manutenzione dei dati. Personalmente ritengo che i trigger siano uno strumento molto versatile, ma da usare con parsimonia poiché inducono ad una naturale proliferazione di procedure di cui è facile perdere il controllo. Inoltre, la loro particolare natura di procedure scatenate in background, talvolta rende il loro comportamento inaspettato agli occhi di un DBA un po' distratto.

Bibliografia

[1] D. Jones – *"The definitive guide to: SQL Server performance optimization"*, Realtimepublishers.com, 2003
[2] D. Esposito – *"Dati integri e controllati con i trigger"*, VBJ N.59, 2004
[3] F. Quaratino – *"Azioni referenziali con SQL Server 2000"*, VBJ N.61, 2005

MDAC 2.8 e Windows 98: problemi di compatibilità



In particolari casi, utilizzando MDAC 2.8 sotto Windows 98, si ottengono blocchi del sistema apparentemente inspiegabili. La soluzione è l'MDAC 2.7!

di Gionata Aladino Canova

Recentemente, abbiamo terminato lo sviluppo di un prodotto con Access XP, utilizzando Office XP Developer per distribuirlo con il runtime. Lo abbiamo installato al secondo cliente dei nostri (potenziali) svariati clienti. Il cliente in questione ha cominciato a chiamare riferendoci di errori per noi atipici. Dopo diverse indagini infruttuose, ci siamo rivolti al supporto tecnico di Microsoft, che ha cominciato a lavorare sul caso. Nel frattempo, abbiamo portato avanti le nostre prove, che ci hanno condotto all'individuazione precisa del problema. Windows 98, in combinazione con alcune stampanti, se l'MDAC 2.8 è installato e se viene utilizzato il VBA, produce svariati errori, alcuni dei quali visibili nella **Figura 1**. Precisiamo che le informazioni che troverete in questo articolo sono frutto di nostri studi, poiché di documentazione ne esiste veramente poca, quindi potrebbero contenere inesattezze. Ma vediamo i fatti in dettaglio.

I passi per riprodurre il problema

Il problema è stato rilevato sia su macchine fisiche che su macchine virtuali, create sotto Microsoft Virtual PC. Per riprodurlo, seguire i passi successivi:

- Installare Windows 98 (su un pc o su una Virtual Machine)
- Installare Office XP con Access o soltanto Access XP (non abbiamo provato con Office 2000 o 2003)
- Installare l'MDAC 2.8
- Installare la stampante *Xerox DocuPrint N17 PS3* o *Brother HL 1450* o *Samsung ML-1520* (solo su un pc reale, poiché è USB); i driver relativi sono reperibili sui siti dei produttori.

Avviare Access e seguire i passi:

- Creare un nuovo database
- Creare un report ed aggiungervi un'etichetta con scritto qualsiasi cosa; salvarlo con il nome "Report1".
- Creare un modulo con il codice

```
Function StampaReport()
  DoCmd.OpenReport "Report1", acViewNormal
End Function
```

- Creare un nuovo pannello comandi da Strumenti \ Utilità Database \ Gestione pannello comandi

Gionata Aladino Canova *programma dai bei tempi del Sinclair Spectrum. Laureato in Informatica, è titolare della Aladino Informatica e socio di TDE Informatica srl. Sviluppa con Microsoft Access, VB.NET e realizza siti in ASP/ASP .NET. Può essere contattato tramite e–mail all'indirizzo info@aladinoinformatica.it*

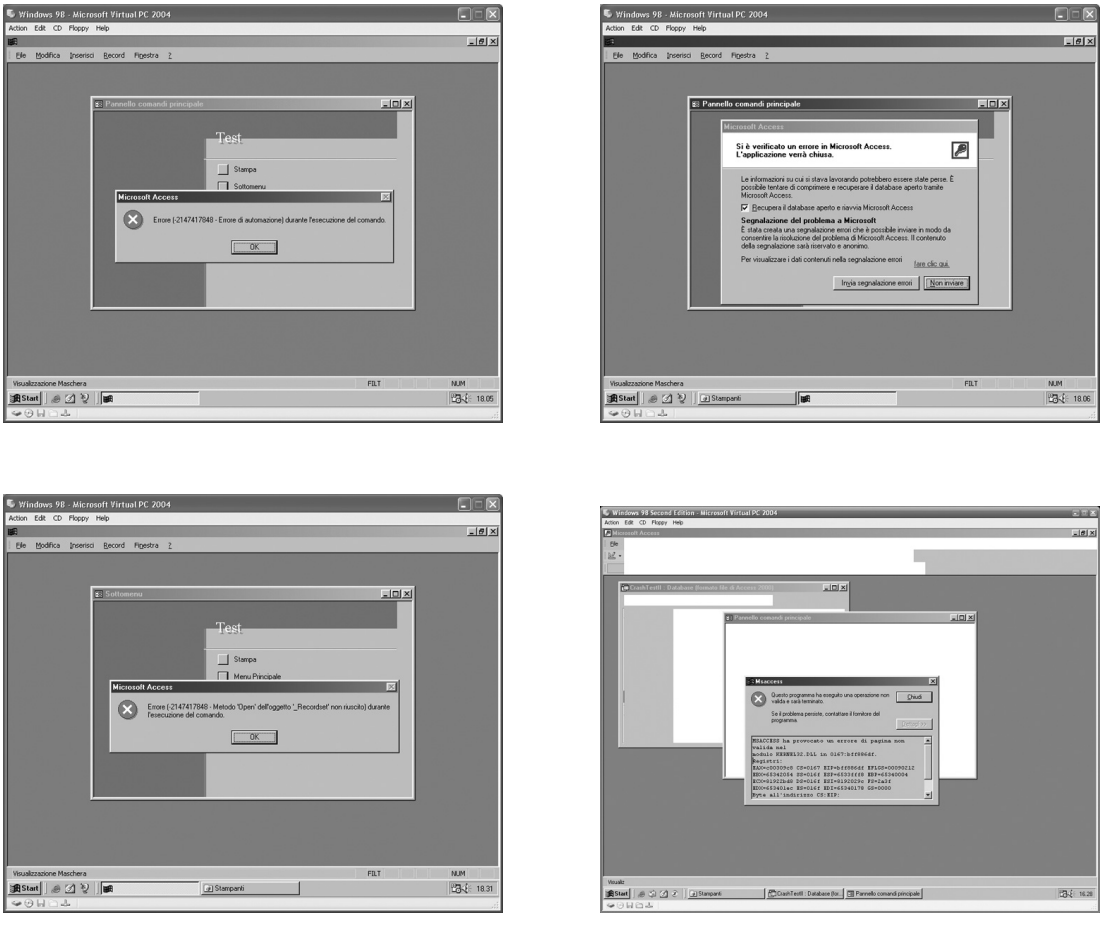


Figura 1 Errori causati dall'accoppiata Windows 98 e MDAC 2.8 con talune stampanti

- Creare un sottomenu con una voce che riporti al menu principale
- Creare nel menu principale, una voce che esegua il codice *StampaReport*
- Creare nel menu principale un richiamo per il sottomenu

Adesso, aprite il pannello dei comandi. Premendo in sequenza il pulsante di stampa e poi quello per il cambio di menu, compariranno i primi errori. In ordine di probabilità, il primo errore a verificarsi dovrebbe essere un *Errore eseguendo il comando*; inserendo nella switchboard un gestore di errori, si vede che l'errore reale è un *Error -2147417848: metodo 'Value' dell'oggetto '_Textbox' non riuscito.*

Cercando sui newsgroup si trova molto poco. Chiudendo e riaprendo il pannello dei comandi, potreste ottenere un *Errore di run-time '-2147417848 (80010108): Metodo 'Open' dell'oggetto '_Recordset' non riuscito*. Insistendo con le prove, in alcuni casi si verificano errori che chiudono brutalmente Access o, addirittura, schermate blu.

Se adesso installate un'altra stampante, con driver forniti insieme a Windows 98, la impostate come predefinita e riprovate ad utilizzare il database, tutto dovrebbe funzionare regolarmente.

Sul sito FTP di Infomedia potete reperire un file già pronto per effettuare le stesse prove.

Analisi del problema

Quando ci si trova in una situazione in cui la documentazione scarseggia e non si hanno punti di riferimento per procedere, una tecnica utile è il *divide et impera*. Si comincia ad eliminare tutto ciò che si può; quando il problema

non si presenta più, forse si ha qualcosa su cui lavorare. Con questo approccio siamo arrivati ad isolare i componenti minimi che scatenano il problema.

Purtroppo non siamo riusciti a trovare nessun fattore comune tra i driver che causano problemi e nessun fattore comune tra quelli che invece funzionano. Infatti, alcuni driver scaricati da internet scatenano il problema, altri no, mentre tutti quelli forniti con Windows 98 che abbiamo provato, sembrano funzionare regolarmente.

Avendo ristretto il problema all'MDAC 2.8, abbiamo ricontattato il supporto tecnico. Purtroppo, quando anche loro hanno riprodotto il problema, al momento di scalare al supporto di livello superiore, sono stati fermati poiché Windows 98 non è più supportato. Tra le varie prove, abbiamo condotto un test anche su Windows ME e sembra che il problema non si presenti.

> Il testing può solo mostrare la presenza di errori, non la loro assenza

Soluzione del problema

Ovviamente la soluzione banale è quella di aggiornare il sistema operativo.

La realtà è che molti clienti non digeriscono questo fatto.

Quindi abbiamo provato con l'MDAC 2.7 e questa strada sembra funzionare. Vediamo i passi da seguire:

1. Se su un PC è già installato l'MDAC 2.8, dobbiamo procedere alla sua disinstallazione. Leggendo [1] scopriamo che il fatto che questa operazione vada a buon fine, è tutt'altro che scontato. Per disinstallare, dopo aver aperto un prompt, passiamo nella cartella *C:\Programmi\File comuni\Microsoft Shared\dasetup* e lanciamo il comando *dasetup /q*. Per eventuali problemi di disinstallazione, consultare [1]
2. Scaricare ed eseguire l'installazione dell'MDAC 2.7, reperibile su [2].

Sviluppare e distribuire con diverse versioni di MDAC

Si pone però un problema. Se, per sviluppare, si utilizza l'ultima versione di MDAC, la 2.8, si ha il problema di doverla cambiare prima di distribuire il pacchetto.

Infatti, copiando un file .mdb che ha riferimenti all'MDAC 2.8 in un ambiente dove sia installata la versione 2.7, otterremo l'errore *Il database o il progetto di Microsoft Access include un riferimento mancante o errato al file 'msado15.dll' versione 2.8*. Vediamo le strade percorribili.

Se utilizzo solo l'ADO e non l'ADOX, posso, nel sistema di sviluppo, selezionare l'ADO 2.7 al posto del 2.8.

Il database portato sotto Windows 98 sembra funzionare correttamente.

Se, oltre all'ADO utilizzo anche l'ADOX, il problema è più complesso.

Infatti, quando inserisco un riferimento all'ADOX, non potendone selezionare la versione, il VBA prende la più recente.

Ossia: se ho installato l'MDAC 2.8, seleziono l'ADO 2.7 e l'ADOX relativo, chiudendo e riaprendo la dialog dei riferimenti, scoprirò che la versione in uso è la 2.7 per l'ADO ma la 2.8 per l'ADOX.

Questo fatto può essere verificato anche tramite un ciclo sulll'insieme References, con la routine

```
Sub StampaVersioniRiferimenti()
  Dim i As Integer
  For i = 1 To References.Count
    Debug.Print References(i).Name & "=" &
References(i).Major & "." & References(i).Minor
  Next
End Sub
```

Compilando il database e spostandolo sotto Windows 98, non otterremo errori per l'ADO ma otterremo l'errore *Il database o il progetto di Microsoft Access include un riferimento mancante o errato al file 'msadox.dll' versione 2.8.* In effetti, il riferimento è corretto, solo la versione è errata.

La soluzione migliore sembra essere di sviluppare con ADO 2.8 e poi copiare il file in un pc con Windows 98 e MDAC 2.7, seguendo i passi:

- Copio il file sotto Windows 98
- Nell'apposita finestra tolgo i riferimenti all'ADO e all'ADOX; la chiudo.

- Riapro la finestra ed aggiungo i riferimenti all'ADO 2.7 e all'ADOX 2.7
- Compilo il database
- Riporto l'MDB sul pc di sviluppo per la distribuzione.

Problemi di riferimenti errati

Purtroppo, agendo nel modo anzidetto, si può verificare un ulteriore problema. Access, ogni volta che compila un oggetto, si memorizza il codice compilato. In alcuni casi è possibile che quel codice abbia dei riferimenti non validi. A quel punto si possono ottenere gli errori più disparati. Uno classico e visibile è vedere una casella di testo di una maschera o di un report, alla quale è associata un'espressione tipo *=Date()*, che visualizza, invece del valore, la scritta *#Nome?*. Cercando sui gruppi con "decompile access problemi" o "decompile access problems" troverete comunque molta documentazione.

Ovviamente, più il database è complesso, più è facile che si verifichi il problema. Ad esempio, noi sviluppiamo utilizzando una libreria .mde aggiuntiva e notiamo con una certa frequenza problemi del genere.

Le soluzioni, sono ancora una volta, molteplici: si può provare ad ovviare con il comando

```
C:\...\MSACCESS.EXE <nomefile>.mdb /decompile
```

che *decompila* il file. Ossia, mette tutti i moduli in stato di non compilato. A questo punto si ricompila il tutto. Purtroppo anche questo approccio può causare problemi. In alcuni casi, ma non siamo mai riusciti a riprodurre un procedimento, abbiamo perso alcune modifiche fatte al codice. Per altri problemi conosciuti, vedere [3].

una tecnica utile è il "divide et impera"

Un approccio che pare funzionare, se si utilizza Visual Source Safe, è quello di rigenerare completamente il database. In pratica, una volta effettuate tutte le modifiche, si archivia il file in Visual Source Safe, si cancella il file in locale e, tramite il comando Strumenti \ SourceSafe \ Create Database From SourceSafe Project... si ricrea il file ex-novo.

A questo punto, *senza compilarlo*, lo si sposta sotto Windows 98 e si procede come già detto. Il file così creato sembra che abbia tutto il codice in stato di decompilato.

Il problema non si pone, infine, se utilizzate un file MDE. Infatti, in questo caso, nel nuovo file che viene creato, il codice viene ricompilato per intero.

Collaudi finali

La nostra applicazione è formata da un file di dati, dall'applicazione e dalla libreria in formato .mde.

Abbiamo ricompilato libreria ed applicazione sotto Windows 98 e MDAC 2.7. Il risultato lo abbiamo riportato sotto Windows XP, nel pacchetto creato in precedenza. Il compilatore, pur sfruttando ADO 2.7 e ADOX 2.8, pare tollerare la situazione. (A tal proposito, se a qualcuno fosse sfuggito, esiste un aggiornamento non automatico per Office XP Dev che risolve tra l'altro alcuni problemi di installazione sotto Windows 98 – vedi [4] e l'SP1 per Office XP – vedi [5]).

Abbiamo poi provato ad installare il pacchetto sotto una macchina virtuale con Windows 98 appena installato, avendo cura di installarvi prima l'MDAC 2.7. Lo abbiamo anche provato sotto Windows ME e XP. I collaudi non hanno fatto rilevare alcun problema anche se, si sa, che il testing può solo mostrare la presenza di errori, non la loro assenza.

Riferimenti

[1] 311720 PRB: Il rollback di MDAC potrebbe non riuscire in Windows 95, Windows 98 e Windows Millennium Edition http://support.microsoft.com/kb/311720/

[2] Download MDAC 2.7 http://tinyurl.com/fyzg oppure http://msdn.microsoft.com/data/downloads/ e cercare MDAC 2.7

[3] Subject: INFO: The real deal on the / Decompile switch http://trigeminal.com/usenet/usenet004.asp

[4] OFFXPDEV: Microsoft Office XP Developer Packaging Wizard Patch Available on MSDN http://support.microsoft.com/default.aspx?scid=kb;en-us;Q305003

[5] OFFXPDEV: Download e installazione di Office XP Developer Service Pack 1 (SP-1) http://support.microsoft.com/kb/313166/it

VBdocman
Documentazione automatizzata del codice

La documentazione del codice semplificata

di Fabio Perrone

La documentazione tecnica di un progetto riveste un aspetto fondamentale; è sufficiente pensare a coloro che scrivono librerie di codice distribuibili a terzi o a chi lavora in team: è sempre la parte più aggiornata di un progetto, a differenza magari dei requisiti o dell'analisi di dettaglio.

> VBdocman presenta senza alcuna ombra di dubbio caratteristiche di completezza ed efficienza

Chi utilizza C# può già servirsi dei commenti XML all'interno del proprio codice sorgente, caratteristica che per i programmatori Visual Basic.NET sarà presente solo dalla versione 2005 di Visual Studio.NET. Tuttavia, la documentazione XML da sola non fornisce un livello d'interfaccia utente avanzato e facilmente leggibile: in questo caso risulta utile ricorrere a strumenti come VBdocman della slovacca Helixoft.

VBdocman è un add-in per Visual Studio .NET, e con l'installazione viene aggiunta un'icona sulla toolbar per un accesso rapido, e una voce di menù sotto Tools.

È importante notare che la documentazione può essere creata in diversi formati: HTML Help 1.x (file con estensione .chm), Help 2 (file con estensione *.HxS, la più recente tecnologia Microsoft per la documentazione, ampiamente utilizzata in Visual Studio .NET), file HTML puri (contenenti solo HTML e Java in modo da distribuire la documentazione immediatamente su un sito Web), file rtf e file XML.

Se si desidera risparmiare tempo senza scrivere nessun commento all'interno del codice, è possibile far sì che VBdocman crei in automatico la documentazione grazie all'attivazione dell'opzione "Compile also non-commented members", che permette di generare i file

Fabio Perrone *si è laureato in matematica e MCP in Visual Basic, lavora presso TSF s.p.a. dove si occupa di progettazione e sviluppo di applicazioni Windows e Web. Può essere contattato tramite e–mail all'indirizzo fperrone@programmers.net.*

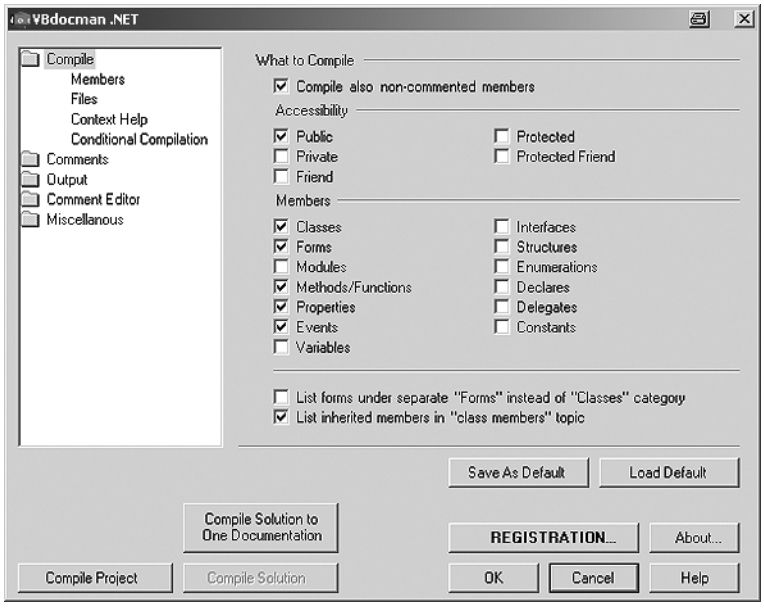


Figura 1 Schermata introduttiva delle opzioni

necessari alla documentazione recuperando le informazioni sui membri delle classi e presentandoli in un formato MSDN-like, quindi completo di link, elenco membri, ecc.

Come è ovvio, la documentazione così creata può non risultare sufficiente se si desiderano informazioni aggiuntive come la descrizione dei parametri passati ai metodi, link a eccezioni o esempi.

In questo caso, è possibile utilizzare i commenti XML utilizzati già in C#, solo che le righe che contraddistinguono i commenti stessi devono essere precedute da ' ' ' o da qualsiasi combinazione di caratteri scelta dall'utente, con la possibilità di utilizzare 5 tag XML personalizzati oltre a quelli già presenti (tipo <summary>, <example>, <param>, ecc.).

Poiché scrivere i tag XML per tutti i membri di un progetto ampio può essere un lavoro particolarmente tedioso, VBdocman aiuta in questa operazione permettendo l'inserimento automatico di commenti XML semplicemente selezionando la porzione di codice da documentare, premendo il tasto destro del mouse e scegliendo "Add VBdocman comment" dal menu contestuale.

Per creare una documentazione più complessa, contenente magari riferimenti ad altri membri, impostazioni dei parametri o altro, dal menu contestuale menzionato in precedenza si può scegliere "Comment editor": avremo a disposizione un vero e proprio editor XML con il quale creare tag XML avanzati che semplificheranno la creazione dei commenti.

VBdocman prevede l'utilizzo di commenti nativi "@-style": sono commenti VB classici con alcune caratteristiche aggiuntive, vale a dire si inizia la riga con il simbolo di commento di VB (') e poi si inseriscono istruzioni che permettono al compilatore della documentazione di identificare la porzione di codice che si sta commentando (@see, @param, ecc.). Per ottenere una documentazione veramente flessibile e completa, è possibile utilizzare due caratteristiche avanzate del programma: i template e le macro.

I primi servono per definire l'output della compilazione; sono forniti template per la generazione dei file .HxS, .chm, .rtf, .html e .xml, completamente ridefinibili in modo da poter personalizzare l'output; le macro sono direttive che sono fornite al compilatore per formattare l'output del testo già direttamente al momento della scrittura dei commenti, utilizzabili però solo nei commenti "@-style". Il notevole vantaggio offerto dalle macro è la possibilità di utilizzare cicli o altre istruzioni proprie dei linguaggi ad alto livello per la creazione di una documentazione tecnica veramente professionale; lo svantaggio è che il loro apprendimento non è immediato e, almeno all'inizio, richiede uno studio approfondito, magari esaminando i template già forniti per comprenderne a fondo il funzionamento.

Per chi sviluppa librerie distribuibili o controlli o per chi lavora in un team di lavoro un aspetto fondamentale riveste il cosiddetto "Context Sensitive Help", in altre parole la possibilità di fornire all'utilizzatore del codice documentato aiuto sui singoli membri con la semplice pressione del tasto F1 posizionandosi sul membro stesso, oppure il tooltip che compare – grazie all'IntelliSense – nel momento della digitazione del codice. Ottenere queste importanti funzionalità con VBdocman è particolarmente semplice: è sufficiente attivare una check-box ed il programma provvederà per noi alla creazione dell'aiuto contestuale.

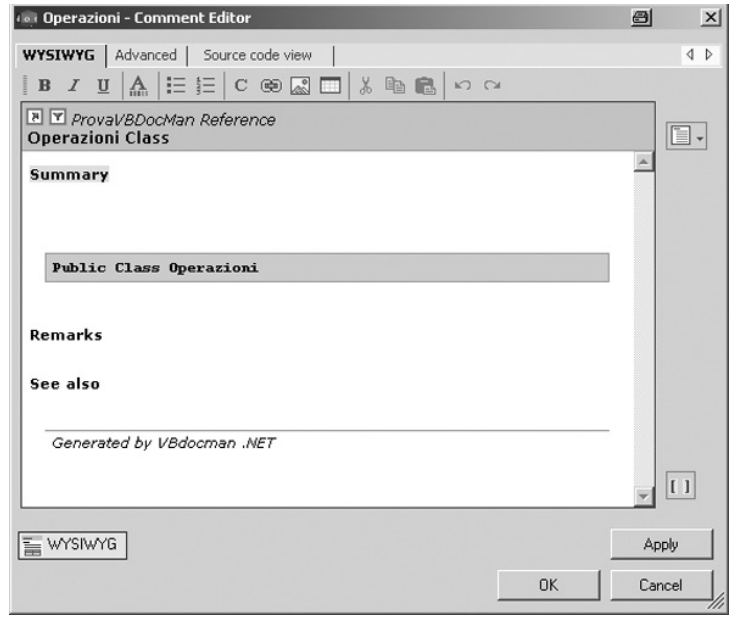


Figura 2 XML Comment Editor

> La documentazione XML da sola non fornisce un livello d'interfaccia utente avanzato e facilmente leggibile

Il programma si presenta in tutta la sua flessibilità nel momento dell'impostazione delle opzioni di compilazione per la creazione della documentazione: è possibile scegliere l'accessibilità dei membri da includere nella documentazione (membri pubblici, privati, friend, ecc.) e quali membri includere (classi, moduli, proprietà, eventi, ecc.). Vale la pena menzionare un'ulteriore opzione di compilazione: la possibilità di compilare una nuova documentazione per l'intera soluzione oppure far sì che ogni progetto facente parte di una soluzione produca la propria documentazione. È ovvio che la prima possibilità è particolarmente valida in soluzioni che condividono namespace tra di loro; la seconda è più adatta se sono soluzioni contenenti progetti di test per librerie distribuibili a terzi.

Conclusioni

La presenza sul mercato di numerosi tool per la documentazione del codice, alcuni anche gratuiti (VBCommenter), evidenziano quanto l'esigenza di produrre documentazione in modo rapido, completo ed efficiente sia sempre più sentita dalla comunità degli sviluppatori. VBdocman presenta senza alcuna ombra di dubbio caratteristiche di completezza ed efficienza; per quanto riguarda la rapidità, la possibilità di inserire documentazione grazie ai template permette di ridurne i tempi di creazione, tuttavia la compilazione di una soluzione complessa richiede una quantità di tempo abbastanza elevata. I prezzi variano dai 143$ per chi acquista da 11 a 20 licenze ai 229$ per la versione monoutente.

C-Sharpener

Un originale add–in per Visual Studio che consente di convertire il codice VB.NET in C#

di Lorenzo Vandoni

Da molti anni, la scelta del linguaggio di programmazione non è più considerata il problema principale da affrontare per lo sviluppo di un'applicazione software.
Nell'ambito del .NET Framework, addirittura, il linguaggio è stato ridotto a una sorta di dialetto al servizio della piattaforma.

Ciononostante ci sono ancora molte situazioni in cui la scelta del linguaggio può risultare importante. Ad esempio, si potrebbe volere sfruttare del codice VB.NET in un progetto C#, oppure si vorrebbe includere un esperto programmatore VB.NET nello sviluppo di un'applicazione C#, senza obbligarlo ad apprendere un nuovo linguaggio.

Da VB.NET a C#

C-Sharpener permette di risolvere questi ed altri problemi. Si tratta di un add-in per Visual Studio, che permette di convertire automaticamente codice VB.NET in C#.

Una volta installato, il programma può essere attivato dal menu *Tools* di Visual Studio. Il processo di traduzione è guidato da un wizard, e consente di selezionare per la traduzione uno dei progetti inclusi nella soluzione attualmente aperta in Visual Studio. Possono essere impostate diverse opzioni, tra cui la possibilità di mantenere i commenti originali, e quella di introdurre operazioni di cast esplicite ove necessarie.

Al termine del processo di traduzione viene mostrata una finestra di riepilogo in cui vengono evidenziati, tra l'altro, il numero di righe di codice convertite e il numero di problemi rilevati durante la traduzione. Questi ultimi vengono suddivisi in due categorie, denominate rispettivamente "transerror" e "transwarning", ed opportunamente evidenziati con commenti all'interno del codice generato.

Il risultato è costituito da un nuovo progetto C#, che viene automaticamente incluso all'interno della soluzione, pronto per essere compilato.

Un piccolo test

La versione di valutazione usata per questa prova consente di tradurre fino a 500 linee di codice, il che mi ha permesso di fare alcune piccole prove di traduzione con tre progetti di ridotte dimensioni – un eseguibile Windows, una semplice DLL, e un'applicazione Internet. La traduzione della DLL non ha provocato nessun problema.

La traduzione dell'applicazione Internet ha dato luogo a 13 "transwarning", relativi principalmente all'introduzione di righe di codice per l'assegnamento esplicito della gestione degli eventi, e all'introduzione di variabili temporanee nei cicli *foreach*. In particolare, questo significa che un'istruzione VB.NET del tipo:

```
Private Sub Page_Init(...) Handles
                                MyBase.Init
```

viene tradotta in C# con un'assegnazione esplicita:

```
base.Init += new System.EventHandler
                          ( Page_Init );
private void Page_Init(...) {
```

Lorenzo Vandoni *si è laureato in Informatica a Milano nel 1990, ed è uno specialista di progettazione e sviluppo con tecniche e linguaggi object–oriented. Ha collaborato alla realizzazione di diversi software commerciali in ambiente Windows. Può essere contattato tramite e–mail all'indirizzo lvandoni@infomedia.it.*

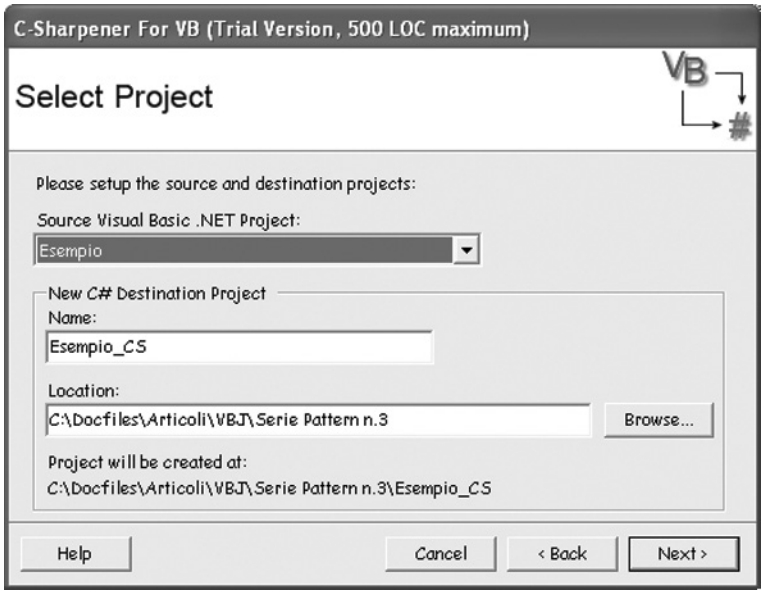


Figura 1 Il wizard di C–Sharpener permette di convertire un progetto VB.NET in C#

e un ciclo del tipo:

```
Dim oObj As MyObject
For Each oObj In MyObjects
oObj.Function()
Next
```

viene tradotto, in modo un po' ridondante, con:

```
MyObject oObj = null;
foreach (MyObject transTemp0 in MyObjects) {
oObj = transTemp0;
      oObj.Update();
}
```

Al di là di queste piccole segnalazioni, però, la traduzione è stata effettuata correttamente, e ho potuto compilare ed eseguire l'applicazione Internet senza nessun problema.

Qualche problema c'è stato, invece, con il programmino Windows che, nonostante abbia dato luogo ad un unico "transwarning", relativo come nel caso precedente all'introduzione della gestione esplicita degli eventi, ha causato un errore di compilazione, dovuto al fatto che è stata "dimenticata" la parola chiave "ref" per passare un parametro per riferimento, come richiesto dalla funzione richiamata. Va detto che questa "dimenticanza" è capitata in un solo caso particolare – l'invocazione di un costruttore della classe base – mentre in altri casi analoghi la traduzione è stata effettuata in modo corretto.

Limitazioni conosciute

Secondo il produttore, C-Sharpener dovrebbe essere in grado di tradurre il 99% del codice VB.NET in C#.

Probabilmente è vero, anche se ci possono essere molti modi con cui questa percentuale potrebbe essere calcolata. Insieme con l'installazione viene fornito un file PDF che riporta l'elenco di tutte le limitazioni conosciute, ovvero dei casi in cui il processo di traduzione potrebbe non funzionare correttamente.

Il documento è piuttosto ampio e dettagliato, e costituisce una lettura interessante, anche perché mette in evidenza molte differenze non banali tra i due linguaggi. Sono subito andato a cercare il caso che mi è capitato nel test, ma non vi ho trovato nessun riferimento, segno che l'elenco non è del tutto esaustivo. Al di là di questa considerazione, però, questo documento può risultare sicuramente utile, anche perché per ogni possibile problema noto viene suggerito un modo per aggirare l'ostacolo.

Altre opzioni

Tra le opzioni che possono essere impostate durante il processo di traduzione vi è la possibilità di mantenere i file intermedi generati durante il processo di traduzione. Si tratta di file XML, che comprendono una completa analisi dei file sorgenti, utilizzata evidentemente come passo intermedio per la traduzione. Anche per progetti semplici, come quelli utilizzati per il test, la dimensione di questi file è sicuramente eccessiva per poter effettuare un'analisi superficiale, ma trovo che aver lasciato la possibilità di accedere a queste informazioni sia molto interessante, anche perché rende potenzialmente implementabili altri tipi di analisi e traduzioni basate su questi file XML, sicuramente più semplici da tratta-

re rispetto ai sorgenti originali. Altra funzionalità interessante è costituita dalla possibilità di lanciare il processo di traduzione dalla linea di comando. Questo permette di utilizzare il tool all'interno di elaborazioni batch, tutt'altro che infrequenti in molte aziende di sviluppo.

Conclusioni

Lo strumento è sicuramente interessante, e non privo di possibili applicazioni pratiche. Tra le altre non va dimenticata la possibilità di portare i propri progetti VB.NET su un sistema open source come Mono, che fornisce un compilatore C#.

Per la valutazione:

Voto complessivo	4
Funzionalità	3
Interfaccia e Usabilità	4
Prestazioni	4
Tempestività sul mercato	4

Ho trovato un po' limitativa la possibilità di tradurre solo interi progetti, e non porzioni di codice, anche perché ho l'abitudine di compilare molte DLL a linea di comando, ma non credo che questa potrà essere vista da molti come una limitazione importante.

Anche l'errore occorso durante la traduzione è piuttosto particolare, e il giudizio complessivo non può che essere positivo.

Scheda prodotto:

Nome e Versione	C-Sharpener
Categoria	Add-in
Produttore (o Distributore)	Elegance technologies
Sistema operativo	Windows 2000 o superiore
Requisiti hardware/software	Nessun particolare requisito
Sito Internet	www.elegancetech.com
e-mail	csharpener_ support@elegancetech.com

Sharp Shooter 1.9

Non solo report: presentiamo uno dei migliori prodotti di reportistica e non solo, interamente pensato per essere integrato con .NET.

di Marco Caridi

"Il generatore più flessibile di report per la piattaforma .NET disponibile sul mercato!". Questo è lo slogan di lancio del prodotto da parte della casa americana 9Rays.NET. Si tratta di una applicazione software specializzata per la produzione di report statistici di ogni genere. La reportistica non è argomento tra i più stuzzicanti, però questo settore ha da sempre conservato la sua nicchia di mercato, traendo vitalità dalla necessità oggettiva di presentare, in modo facile e flessibile, risultati di operazioni sui dati. Il prodotto, già al primo impatto, sembra essere veramente di grande semplicità di uso e potenzialità. Completamente compatibile, o per meglio dire, "pensato" per il framework .NET, Sharp Shooter facilita drasticamente la generazioni di report.

È un sistema aperto, consente personalizzazioni grazie alla possibilità di estendere qualsiasi funzione già presente con procedure software proprietarie sviluppabili n VB.NET o altri linguaggi. Per esempio è possibile sviluppare filtri di esportazione sotto forma di Dynamic Link Libraries e definirli nell'ambiente di reportistica come plug-in esterni. Grazie alle sue funzioni di import ed export, per esempio, è possibile importare namespaces, variabili locali, procedure, controllando nel dettaglio l'intero processo di produzione del report, nonchè importare vecchi report generati dal tool Crystal Reports. Uno degli effetti grafici più accattivanti è la possibilità di utilizzare formati ed orientamenti testuali e grafici differenziati nella stessa pagina del report. Il prodotto è "ovviamente" basato su una logica ad oggetti gerarchica per cui qualunque estensione applicativa può essere salvata in una libreria e riutilizzata in applicazioni future. Entriamo nel merito dell'architettura applicativa per meglio chiarirne gli aspetti funzionali. La definizione di un report è basata sul concetto di modello (o *template* per gli estimatori della lingua inglese). Selezionato dalla libreria un modello tra quelli forniti con il software o di propria personalizzazione, l'engine provvede a produrre i report. Questo "core" applicativo accede ai datastore per leggere e selezionare i dati e produce un metadato (in formato XML) pari al report desiderato. L'ultimo stadio software è costituito dal "presentation layer". Grazie al concetto di stile in connubio con XML stesso, il metadato è presentato all'utente nello stile e nella forma prescelti. In particolare è possibile filtrare il metadato producendo un report nei seguenti formati:

- (X)HTML – per una facile pubblicazione web;
- TESTO – per meglio trattarlo e trasferirlo in sistemi monotematici come Unix e similari;
- IMMAGINE – in un formato selezionabile tra moltissime opzioni (jpg, bmp, gif, tiff…);
- PDF – per renderlo adattato allo strumento Adobe Acrobat;
- RTF;
- altri ancora.

Si è parlato di piena compatibilità con il framework .NET, ma cosa implica questo per l'utente esperto? È garanzia di estendibilità del software mediante procedure sviluppate con uno qualsiasi dei linguaggi di scripting offerti dalla piattaforma. Il modello dati gerarchico caratteristico di ADO.NET si sposa perfettamente con Sharp Shooter consentendo facilmente di inserire nei report i

***Marco Caridi** è laureato in Ingegneria Elettronica presso la Università` "La Sapienza" di Roma. Si occupa di Analisi e Sviluppo Software. Da diversi anni è consulente di progetto presso il centro servizi VAS di TIM in Roma. Può essere contattato tramite e–mail all'indirizzo mcaridi@infomedia.it.*

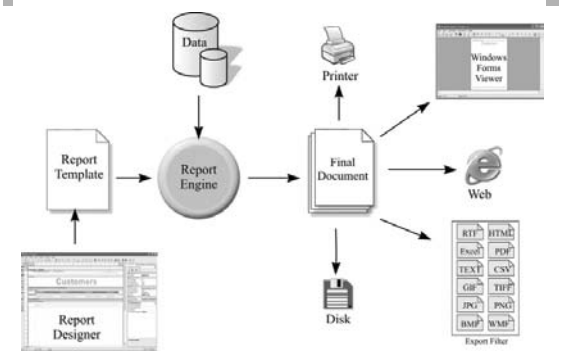


Figura 1 Visione schematica dell'uso del prodotto

dati presi dall'oggetto. Anche ASP.NET si presenta al tool con un biglietto da visita più che di riguardo. Infatti l'integrazione completa tra i due ambienti consente di creare e pubblicare facilmente report su Internet Information Server. Il tool non trascura nemmeno elementi di corredo come le intestazioni di pagina, i numeri di pagina automaticamente generati, logo, e qualunque info desideriate visualizzare all'interno del report stesso rendendolo piacevole e facile da leggere. Una nota di merito riguarda la fornita raccolta d'esempi (oltre 90) inclusa nel pacchetto software. Esempi di personalizzazioni e sviluppo di filtri ad hoc sia in VB.NET sia in DELPHI.NET. Ce n'è insomma per tutti i gusti! Prima di chiudere questa recensione veniamo agli aspetti commerciali. Il prodotto può essere scelto tra quattro possibili versioni che vanno dalla Professional alla Light. Con la prima, la casa fornisce persino il codice sorgente dell'applicazione! Il requisito minimo è che il framework.NET sia installato sulla macchina dell'utilizzatore finale del software. Questa breve recensione non vuole e non può sostituirsi alla documentazione ufficiale alla quale si rimanda per approfondimenti del caso.

Scheda prodotto:

Nome e Versione	Report Sharp Shooter 1.9
Categoria	Reporter
Produttore (o Distributore)	9Rays.Net – www.9rays.net
Sistema operativo	Windows
Requisiti hardware/software	.NET
Prezzo	A partire da 375 USD

Ottimizzare i lock sui database

Quattro cose da ricordare con SQL Server

di Ingo Rammer

Come sviluppatori, talvolta tendiamo a vedere i database come stupidi mezzi per memorizzare informazioni con un'unica responsabilità: memorizzare i dati dell'applicazione in modo persistente e recuperarli il più rapidamente possibile. Una attenta attività di tuning e di profiling è un compito che gran parte di noi può lasciare agli esperti di database e agli amministratori.

Ma se ci si imbatte in problemi di timeout, deadlock e prolungati lock di transazioni, un rapido sguardo dietro le quinte può essere più che proficuo. Il principale problema dei database, e al contempo quasi il motivo per cui esistono, è il fatto che più utenti possono accedere alla stessa risorsa contemporaneamente, richiedendo operazioni sia di lettura sia di scrittura.

Per mantenere i dati consistenti, due principali sistemi differenti si sono consolidati per i database relazionali: l'utilizzo dei lock di database (locking) e il versioning dei resultset (rowset versioning). Ad esempio, SQL Server (fino alla versione 2000 compresa) utilizza il primo metodo e introduce l'altro nella versione 2005.

D'altro canto, i database Oracle evitano la creazione di lock di database per gli accessi a sola lettura (SELECT SQL) e utilizzano invece il versioning dei rowset (semplificando, ciò significa che il database mantiene una copia dei dati selezionati per ciascun utente per tutta la durata della transazione).

Locking significa che vengono creati e controllati differenti tipi di lock durante la durata della transazione.

Questi lock possono limitare l'accesso a un database, a una tabella, a una pagina, a un intervallo (di record o di criteri di selezione) o a singoli record.

La creazione e il controllo dei lock dipende dal livello di isolamento della transazione selezionata, il che porta a una matrice di compatibilità strettamente definita tra operazioni concorrenti. I tipi principali di lock sono "S" e "X", dove S significa shared e X si riferisce a un lock esclusivo.

Ad esempio, se la transazione A vuole modificare gli stessi blocchi di informazione sottoposti a lock dalla transazione B, dovrebbe attendere finché il lock non viene rimosso. Ciò avviene normalmente all'atto del COMMIT o del ROLLBACK.

Finora, tutto semplice...

Per verificare la necessità e l'effetto dei differenti tipi di lock, si può selezionare uno dei livelli di isolamento della transazione:

thinktecture

***Ingo Rammer** è il fondatore di thinktecture, una compagnia che aiuta gli sviluppatori e gli architetti software a progettare e implementare applicazioni e Web service .NET. Partecipa come speaker alle conferenze internazionali dedicate a tali argomenti ed è autore del best–seller Advanced .NET Remoting (APress). È Microsoft Regional Director per l'Austria e può essere contattato tramite il sito http://www.thinktecture.com/staff/ingo.*

- Read Committed: vengono letti i dati sottoposti a commit;
- Read Uncommitted: vengono letti i dati delle transazioni correntemente in esecuzione;
- Repeatable Read: i record interessati dalla SELECT verranno sottoposti a lock. Per le altre transazioni: nessun UPDATE possibile, solo INSERT;
- Serializable: i record e gli intervalli interessati dalla SELECT verranno sottoposti a lock. Per le altre transazioni: nessun UPDATE possibile e nessun INSERT possibile nei criteri di selezione della clausola WHERE.

Ma che significa?

In poche parole, ciò significa che i lock inutilmente estesi o duraturi distruggono le prestazioni e la scalabilità delle applicazioni.

Significa anche che i lock troppo piccoli distruggono i dati. Per garantire la stabilità dell'applicazione, è estremamente importante che lo sviluppatore conosca il comportamento di locking del proprio database e delle proprie applicazioni.

Passo 1: Analizzare i deadlock e i blocchi

In un modo o nell'altro, gran parte di noi avrà già visto (o causato) deadlock nelle applicazioni database.

Questi deadlock vengono causati da due transazioni che cercano di accedere contemporaneamente alle stesse due risorse (o in effetti, agli stessi due lock) ma in ordine inverso.

Ad esempio, se la transazione T1 pone un lock sull'oggetto A e poi prova ad accedere all'oggetto B mentre, al contempo, la transazione T2 pone un lock sull'oggetto B e tenta di accedere all'oggetto A. In questo caso, nessuna delle transazioni sarà in grado di terminare.

Per evitare questi tipi di deadlock, di solito è sufficiente far sì che tutte le applicazioni accedano ai dati nella stessa sequenza. Se entrambe le transazioni T1 e T2 accedono prima alla tabella A e poi alla tabella B, non provocheranno alcun deadlock (poiché il lock su A verrà rimosso dopo che la prima transazione termina, in modo che la seconda transazione possa accedere allo stesso oggetto). Se l'applicazione deve gestire i deadlock, si è "fortunati" finché li si possono ignorare.

Tutti i sistemi di database attualmente più diffusi riconoscono questi deadlock, designano una delle transazioni come vittima del deadlock e forzano l'abort e il rollback di questa transazione. Per la propria applicazione, ciò di solito provocherà la generazione di una eccezione che permette di riavviare la transazione. I blocchi normali (se una transazione deve semplicemente attendere un po' perché un lock sia rimosso) sono più difficili da riconoscere poiché incidono solo sul throughput dell'applicazione – in transazioni al secondo.

Consideriamo nuovamente l'esempio precedente: se T1 e T2 utilizzano gli oggetti A e B nella stessa sequenza, ma T1 decide di porre un lock su B per alcuni secondi (ad esempio, durante una transazione più grande), allora T2 dovrà attendere finché T1 rilascia i lock.

Questa situazione è detta di blocco. Il problema con questi blocchi è che possono seriamente incidere sulla scalabilità dell'applicazione mentre si verificano talvolta su tabelle apparentemente non importanti.

Un problema di blocking molto comune, ad esempio, si ha quando una singola tabella viene utilizzata come base per i valori identity o per le chiavi primarie (invece di utilizzare le colonne IDENTITY di SQL Server).

In questi casi si può osservare una tabella, spesso denominata in modo simile a "COUNTERS", con campi come "COUNTER_NAME" e "NEXT_VALUE".

Questa tabella conterrà un record per ciascuna tabella di dati, per cui verrà utilizzata per fornire gli identificatori della chiave primaria.

Prima di eseguire l'INSERT di un record in una tabella dell'applicazione, si dovrà prima selezionare il valore NEXT_VALUE corrente, incrementarlo ed eseguire l'UPDATE della tabella COUNTERS prima di eseguire l'INSERT effettivo.

Operazioni come queste possono avere un effetto collaterale imprevisto: tutti gli INSERT verranno eseguiti serialmente in una transazione. T2 può solo iniziare l'inserimento in qualsiasi tabella dopo che T1 ha eseguito il commit. In tal modo, si trasforma effettivamente SQL Server in un database single-user, perlomeno per quanto concerne le operazioni di INSERT (per inciso: invece di seguire l'approccio descritto, ad esempio, si possono utilizzare le colonne IDENTITY in SQL Server e effettuare una invocazione "SELECT SCOPE_IDENTITY()" dopo l'operazione di INSERT se si deve accedere alla chiave primaria appena

generata. In alternativa, si possono utilizzare valori GUID generati dal client.

Se si deve realmente ricorrere a tabelle COUNTER come quella mostrata prima, può essere una buona idea recuperare un gruppo di ID distinti – ad esempio 100 – in una transazione separata e conservare questi ID sul client). Per identificare un comportamento di blocco come quello descritto prima, consiglio di familiarizzare con SQL Profiler.

In tal caso, si può avviare il profiler e definire un filtro come "Duration > 1000" per richiedere i dati di log di tutte le transazioni che hanno dovuto attendere più di un secondo per il rilascio di un determinato lock.

> Locking significa che vengono creati e controllati differenti tipi di lock durante la durata della transazione

Passo 2: Pensare come SQL Server

Si può anche sfruttare una caratteristica pressoché sconosciuta del motore di SQL Server: in gran parte dei casi SQL Server porrà un lock solo su ciò che esattamente è stato modificato. Anche se ciò sembra molto banale, in alcuni casi questo unico fatto può far accrescere la scalabilità della propria applicazione.

Basta immaginare di avere una tabella "Products" che contiene, tra gli altri, i campi ProductID, UnitPrice e UnitsInStock (per inciso: combinare il livello attuale di giacenza di magazzino con i dati statici di un prodotto può non essere una best practice. Ma l'ho visto fare abbastanza spesso, pertanto ritengo sia un esempio molto valido in questo caso. Inoltre, si tenga conto che sto utilizzando delle istruzioni SQL solo per semplificare la visualizzazione. Nei sistemi reali, normalmente si vedrebbero delle query con parametri o delle invocazioni a stored procedure).

Se si utilizzasse questa tabella per registrare una consegna effettuata da uno dei propri fornitori, si eseguirebbe molto probabilmente una sequenza di statement simili a "UDPATE products SET UnitsInStock = UnitsInStock + 10 WHERE ProductID = 7". Se un ulteriore processo richiedesse contemporaneamente i dati (prima del COMMIT), ad esempio inviando uno statement come "SELECT UnitPrice FROM Products WHERE ProductID = 7", ciò provocherebbe il blocco del secondo processo.

È interessante notare che ciò non sarebbe necessario, poiché il campo UnitPrice non viene modificato dalle precedenti istruzioni UPDATE. È qui che entra in gioco la caratteristica di SQL Server di cui ho detto precedentemente. Ad esempio, si può creare un indice non-unique, non-clustered che contiene i campi ProductID e UnitPrice.

In questo caso, la seconda richiesta non verrebbe bloccata, poiché la query potrebbe essere pienamente soddisfatta utilizzando l'indice senza dover ricercare i dati originali nella riga sottostante. Siccome l'indice non è stato modificato dall'aggiornamento, non verrà bloccato, pertanto la seconda richiesta può essere servita immediatamente.

Passo 3: Attenzione agli aggiornamenti

Se seguite da un po' di tempo la rubrica Architect's Corner, saprete già che non ritengo necessariamente che i metodi di accesso a database generati automaticamente siano una grande idea nelle parti delle proprie applicazioni che devono supportare un notevole carico di transazioni in un ambiente scalabile.

Un ulteriore motivo di questa convinzione è il seguente: basta immaginare che il processo di consegna illustrato prima non verrà gestito con una semplice istruzione di UPDATE ma invece avverrà utilizzando un DataSet, o un O/R Mapper o un metodo simile.

In questo caso, è facilmente possibile che, dopo una prima istruzione SELECT, possa essere generato un comando SQL come il seguente: "UDPATE products SET UnitsInStock = 46, UnitPrice = 97 /* ... altre colonne ... */ WHERE ProductID = 7". In questo caso, il metodo comprende tutte le colonne nella istruzione UPDATE (e non solo l'unica colonna modificata che sarebbe stata sufficiente).

L'ottimizzazione mostrata al passo 2 (che permetteva di utilizzare un indice separato per accrescere il grado di parallelismo) non funziona più in questo caso.

Statement come questo fanno sì che venga automaticamente generato un lock per ciascun indice nelle righe interessate e perciò influisce in modo negativo sulla scalabilità dell'applicazione anche quando non sarebbe necessario.

Step 4: Suggerimenti di locking come ultima risorsa

Fino alla versione 6.5 di SQL Server, il lock più piccolo possibile era a livello di pagina di database che può contenere alcuni record (di solito, una pagina è di 4KB).

Dalla versione 7.0 di SQL Server, il motore del database decide se è meglio porre un lock a livello di pagina o a livello di record.

In alcuni casi può essere perciò utile evitare la creazione di lock bloccanti a livello di pagina, suggerendo esplicitamente a SQL Server di utilizzare lock a livello di record.

Si può farlo utilizzando il suffisso "with (ROWLOCK)" come in questo caso: "UDPATE products WITH (ROWLOCK) SET UnitsInStock = UnitsInStock + 10 WHERE ProductID = 7.

Conclusioni

Come si è visto, può essere necessario trattare SQL Server come ben più di un semplice deposito di dati per evitare i deadlock e i lock di lunga durata.

Per aumentare, o in primo luogo abilitare, la scalabilità delle proprie applicazioni, si deve anche conoscere il comportamento del locking, e le relative conseguenze, negli scenari di utilizzo concorrente. Ovviamente, nello spazio dedicato a questa rubrica, ho potuto fornire solo alcuni spunti su questo argomento. Perciò suggerisco di investire del tempo con SQL Profiler per tracciare gli effetti dei differenti livelli di isolamento delle transazioni in differenti scenari, in modo da imparare a stimare il comportamento di locking del proprio database.

Inoltre, vi raccomando due libri. Anche se di solito sono considerati letteratura per amministratori di sistema, credo realmente che ogni sviluppatore di applicazioni database critiche debba leggere o utilizzare come riferimento i testi [1] e [2]. Facendolo, la lotta contro deadlock o timeout bloccanti diverrà un ricordo del passato.

Riferimenti

[1] Ken England "Microsoft SQL Server 2000 Performance Optimization and Tuning Handbook", Elsevier Digital Press

[2] Kalen Delaney "Inside Microsoft SQL Server 2000", Microsoft Press

Dieci tecniche per controllare lo spam

CipherTrust rivela la propria top ten dei suggerimenti per controllare e combattere il frustrante afflusso di spam nelle caselle aziendali.

di David Stanley

In un'era in cui si tende sempre più verso una "e-economy", è di capitale importanza che l'industria delle telecomunicazioni si adatti a questo scenario. Vista la rapida migrazione verso l'utilizzo di dispositivi mobili al posto delle linee cablate e la crescente quantità di abitazioni con accesso a Internet, si potrebbe immaginare che le aziende leader stiano orientandosi verso questa situazione. Ma così non è. Un buon numero delle aziende dell'indice FTSE 100 non possono neanche essere raggiunte via email e ciò può essere causato solo da una cosa: la minaccia dello spam. Lo spam in azienda si è indubbiamente trasformato da fastidio a problema critico. Nessun approccio allo spam può funzionare da solo, e nessuna singola tecnologia è in grado di contrastarne l'aggressione. Ritengo però che utilizzando le seguenti dieci tecniche si possano avere buone armi contro la minaccia dello spam.

1. *Diversità – L'approccio a cocktail*
L'identificazione è il primo passo verso il blocco dello spam; a differenza dei virus non vi è alcuna unica soluzione che blocchi tutto lo spam. Ciò che è necessario è un "approccio a cocktail" in cui le aziende impiegano più tecniche, comprese le analisi euristiche e i tool di filtraggio collaborativo in tempo reale.

2. *Flessibilità – Misure differenti per persone differenti*
Un problema comune nelle aziende è la diversa definizione di cosa sia esattamente lo spam. Per una organizzazione l'invio massiccio di email può essere visto come un fastidio, mentre per altre è un'operazione essenziale. Le soluzioni antispam devono permettere agli amministratori di imporre regole differenti e anche permettere di applicare regole differenti a utenti differenti. In teoria, una soluzione antispam aziendale comprenderà un gestore integrato di politiche, che impone la politica aziendale all'intero sistema di email e permette insiemi differenti di regole per differenti utenti e gruppi.

3. *Expertise – Conoscere il nemico*
Gli spammer stanno costantemente migliorando i propri metodi, particolarmente man mano che le aziende hanno infine iniziato a difendersi. I produttori devono essere in grado di sviluppare e distribuire policy, firme, parole chiave e valori alle aziende utilizzando la propria soluzione. Solo attraverso un costante miglioramento, una soluzione può continuare ad essere responsiva agli spammer anche in presenza di nuove minacce.

4. *Autenticazione – Posso vedere un documento?*
Gli spammer investono una notevole quantità di tempo e di impegno nel celare la propria identità e il punto di origine dei loro attacchi. Fortunatamente, ciò lascia segnali rivelatori alle spalle. Un buon tool deve essere in grado di autenticare l'indirizzo DNS del server che sta inviando il messaggio. Se il DNS Lookup inverso non riesce ad autenticare il dominio di una connessione in ingresso, ciò può indicare un server dirottato, e ciò può rappresentare un dato prezioso per identificare lo spam.

***David Stanley** è VP e MD, EMEA, di CipherTrust.*

5. *Collaborazione – Uniti resistiamo, divisi cadiamo*
Le aziende devono sfruttare gli sforzi collaborativi della comunità Internet per poter definire nuove signature e politiche. Non facendolo, si rende incompleta ogni soluzione. I migliori produttori hanno una stretta collaborazione con i principali ricercatori e con le iniziative collaborative per assicurare risposte aggiornate e importanti alle minacce dello spam.

6. *Apprendimento – Se mi inganni una volta, è colpa tua. Se mi inganni due volte, è colpa mia*
Gli spammer sono implacabili. Con le raffiche di email che costano quasi nulla, hanno ogni motivo per lanciare ripetutamente lo stesso attacco. Per quanto frustrante sia ricevere spam, lo è ancor più ricevere lo stesso spam all'infinito. Nuove regole devono essere create automaticamente man mano che nuove minacce emergono per impedire in futuro uno spam analogo e/o permettere agli utenti finali di essere assistiti nella intercettazione dello spam.

7. *Verifica – Controlla da te*
Le aziende devono autorizzare i dipendenti a esaminare e inserire i messaggi nelle proprie code di quarantena, pur rispettando la supervisione complessiva dell'amministratore. Una sfida importante per gli amministratori incaricati delle soluzioni aziendali anti-spam è la gestione delle aspettative e delle preoccupazioni degli utenti finali. Dopo l'introduzione della soluzione anti-spam, gli utenti finali avranno timore che vengano bloccate le mail legittime. Una vera soluzione aziendale deve comprendere tool che permettano agli amministratori di fornire l'accesso a code di quarantena per alcuni utenti o per tutti gli utenti, permettendo loro di essere tranquilli sui messaggi che sono stati bloccati.

8. Automazione – Il miglior amico dell'amministratore
Conseguire e mantenere alti tassi di blocco anti-spam con un basso numero di falsi positivi è un impegno costante. Per garantire che gli amministratori non siano costretti a investire troppo tempo in questo compito, una soluzione anti-spam deve essere in grado di mantenere una efficienza indipendentemente dall'intervento dell'amministratore. Una generazione automatica di regole, dove le regole vengono create senza l'intervento dell'amministratore e l'inserimento in whitelist degli utenti sicuri, migliorerà "al volo" il tasso di rilevazione e farà diminuire i falsi positivi.

9. *Sicurezza – Osservare il quadro complessivo*
È bene proteggere l'intero sistema email dagli attacchi basati su email. L'intero sistema di email è un obiettivo non solo per gli spammer, ma anche per hacker e intrusi. Anche gli spammer sono hacker, che principalmente raccolgono indirizzi email su mailserver e gateway. Un buon sistema aziendale di email deve tener conto di queste vulnerabilità ed essere in grado di proteggere almeno sé stesso e in teoria l'intero sistema di email da questi attacchi.

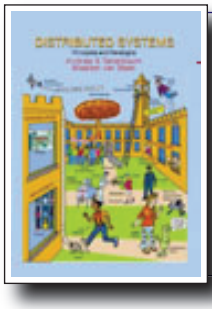
10. *Profiling – L'immondizia fuori, la comunicazione entra*
La sfida in corso delle aziende per combattere lo spam è il compromesso tra un alto tasso di rilevazione e un alto tasso di falsi positivi. Fino a poco tempo fa, questa relazione era fissa. Man mano che aumentava il tasso di rilevazione, aumentava anche il numero di falsi positivi. L'unico modo per rompere questo modello, conseguendo un alto tasso di rilevazione e minimizzando i falsi positivi, è installare una soluzione che può prendere decisioni complesse e dalle molte sfaccettature. Utilizzando un sistema di profiling, gli amministratori possono adottare in modo aggressivo il blocco dello spam senza il rischio di perdere mail legittime.

Una esauriente soluzione anti-spam

Amministrare queste dieci tecniche per controllare lo spam, può diventare rapidamente un compito ingestibile. Oggigiorno, le aziende stanno cercando di prevenire lo spam utilizzando solo una o due di queste tecniche, il che produce una scarsa rilevazione e un alto tasso di falsi positivi. Una soluzione completa dovrebbe basarsi su tutti i principi di cui si è discusso. Fornendo questa protezione sul gateway, in una piattaforma irrobustita, resistente agli attacchi, si migliorerà la sicurezza complessiva dell'azienda.

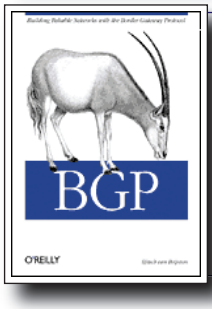
CipherTrust espone a Infosecurity Europe 2005, l'evento principale in Europa per l'Information Security. Giunta al suo decimo anniversario, Infosecurity Europe comprende seminari, presenta nuovi prodotti e servizi, conta oltre 250 espositori e 10.000 visitatori. Si tiene nel mese di Aprile al Grand Hall, Olympia, Londra. Informazioni a: www.infosec.co.uk

IN OFFERTA VBJ 63



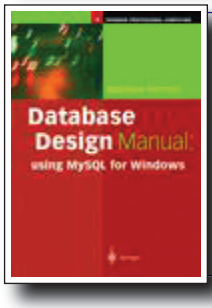
Distributed Systems: Principles and Paradigms (International Edition)
di A. Tanenbaum e M. van Steen

Prentice Hall
ISBN 0131217860
803 pp – 69,95 €



BGP
di I. van Beijnum

O'Reilly
ISBN 0596002548
288 pp – 41,40 €



Database Design Manual: using MySQL for Windows
di Norman, M.

Springer
ISBN 1852337168
225 pp – 69,50 €

Scrivi a book@infomedia.it specificando nell'oggetto della e-mail:

IN OFFERTA VBJ n. 63

OPPURE inviaci il coupon sottostante al numero di fax

0587/732232

Potrai acquistare i libri qui riportati con uno

SCONTO ECCEZIONALE

del **10%** anche se acquisti solo **un libro**

OPPURE

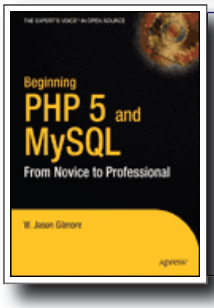
del **20%** se acquisti **3 libri**

Effective XML–50 Specific Ways to Improve Your XML
di Elliotte Harold

Addison Wesley
ISBN 0321150406
336 pp – 46,95 €

Sistemi di basi di dati Fondamenti 4/E
R. Elmasri – S. Navathe

Pearson Italia
ISBN 8871922204
580 pp – 35,00 €



Beginning PHP 5 and MySQL: From Novice to Professional
di Gilmore W. J.

Apress
ISBN 1893115518
736 pp – 40,50 €

GRUPPO EDITORIALE INFOMEDIA

Web: http://book.infomedia.it • E-mail: book@infomedia.it • Fax: 0587/732232

THE SOFTWARE SIDE OF YOUR MIND — VBJ 63

DATI ANAGRAFICI

NOME	COGNOME	CODICE CLIENTE	
VIA/PIAZZA	CAP	CITTÀ	PROV.
TELEFONO (*) FAX	E-MAIL (*)		
DITTA			
INTESTAZIONE FATTURA		P.IVA	

Garanzia di riservatezza - Gruppo Editoriale Infomedia garantisce la massima riservatezza dei dati da lei forniti e la possibilità di richiederne gratuitamente la rettifica o la cancellazione scrivendo a: Responsabile Dati Gruppo Editoriale Infomedia Srl - v. Valdera P. 116 - 56038 Ponsacco (PI). Le informazioni custodite nel nostro archivio elettronico verranno utilizzate al solo scopo di inviarle proposte commerciali. In conformità alla legge 675/96 sulla tutela dati personali.

ORDINE

TITOLO	CODICE ISBN	PREZZO	QUANTITÀ

(*) campo obbligatorio

SPESE DI SPEDIZIONE

☐ PER POSTA - + €3,00 ☐ A MEZZO CORRIERE - euro 7,00

TIPO DI PAGAMENTO

☐ ALLEGO ASSEGNO BANCARIO INTESTATO A "GRUPPO EDITORIALE INFOMEDIA Srl" - NON TRASFERIBILE

☐ CONTRASSEGNO (solo per spedizione a mezzo posta + €3,00)

☐ ALLEGO VERSAMENTO SU CC. POSTALE N.14291561 INTESTATO A "GRUPPO EDITORIALE INFOMEDIA SRL - PONSACCO". Sul modulo di C.C.POSTALE scrivere la causale del versamento.

☐ ALLEGO FOTOCOPIA DEL BONIFICO BANCARIO EFFETTUATO SU C/C 10005424 CAB 71120 ABI 6370 CASSA DI RISPARMIO DI VOLTERRA AGENZIA DI PONSACCO

☐ CARTA DI CREDITO VISA/MASTERCARD/CARTASÌ

N. _____ SCADENZA _____

TITOLARE _____

FIRMA _____ NATO IL _____

Le spese di spedizione sono soggette a variazioni in base alle tariffe postali. Per i pagamenti con assegno, c/c postale e bonifico bancario consigliamo di chiedere conferma in redazione oppure di consultare il nostro sito internet. Grazie.

LIBRI

Photoshop CS a portata di mano

Autore	J. C. Teague, W. Dietrich
Editore	Mc Graw Hill
ISBN	8835643954
Lingua	Italiano
Anno	2004
Pagine	624
Prezzo	€ 38,50

Questo libro è rivolto a quella fascia di persone che vuole imparare Adobe Photoshop da zero o comunque da un'esperienza non approfondita, e a chi vuole comprendere velocemente le novità introdotte rispetto alla versione precedente. Si comincia ovviamente con un'attenta analisi dell'interfaccia di Photoshop, che a primo impatto può sembrare scarna, mentre in realtà cela una mole di comandi impressionante, spesso ignorati anche da chi lo utilizza quotidianamente da anni. La seconda delle cinque parti, in cui è stato giustamente diviso il libro per identificare i vari "temi" trattati, riguarda l'uso degli strumenti disponibili e dei principali comandi. Seguono le attività legate alla fotografia digitale ed al video, la stampa con annessa la gestione del colore ed infine il Web.

Ogni "parte" contiene diversi paragrafi, tutti abbastanza scorrevoli, grazie ad una descrizione sapiente e mai macchinosa dei comandi da eseguire, appoggiata da un buon numero di immagini "pratiche", che fanno riferimento a quanto spiegato dal testo. Tra un paragrafo e l'altro si possono trovare riferimenti ai comandi da tastiera, utilizzabili per richiamare velocemente lo strumento o la funzione di cui si sta parlando, con la giusta combinazione sia per Windows che per Mac. Oltre questo, si incontrano spesso box grigi e testi contrassegnati dalle scritte "note" e "suggerimento", contenenti informazioni utili alla comprensione e l'applicazione del capitolo; in alcuni casi, i box grigi parlano di persone famose nel loro campo, di come utilizzano tal strumento o funzione, con relativi indirizzi Web per approfondire meglio. Nel complesso, il libro riesce a trattare tutti gli argomenti possibili o gli manca poco per farlo, ovviamente questo penalizza quelli più difficili, come la stampa tipografica e la gestione del colore, che per essere trattati a fondo richiederebbe un libro di pari dimensione. Concludendo, un libro relativamente economico per la sua corposità, facile da leggere e ben strutturato, se pensate che Photoshop farà parte del vostro futuro, è una lettura che mi sento di consigliare senza remore.

Pro

Semplice da leggere, offre una buona panoramica di tutti gli strumenti ed i maggiori comandi.

Contro

Non copre approfondimentalmente argomenti difficili come la stampa tipografica e la gestione del colore.

Stefano Arcidiacono

Computer a responsabilità limitata

Autore	David Harel
Editore	Einaudi
ISBN	8806160095
Lingua	Italiano
Anno	2002
Pagine	196
Prezzo	€ 13,00

Un piccolo classico, questo di David Harel, *computer scientist* di fama planetaria, che divulga con brio concetti applicativi di algoritmica e complessità computazionale per mostrare matematicamente, non empiricamente o filosoficamente, "dove le macchine non riescono ad arrivare", come recita il sottotitolo. Punto di partenza: non tutti i problemi sono alla portata dei calcolatori, non importa quanto questi siano potenti o programmati in linguaggi avanzati. Esistono problemi *indecidibili* (o non computabili) per i quali non c'è algoritmo di risoluzione; e problemi *decidibili* (computabili), ma con algoritmi di risoluzione troppo onerosi. Sono indecidibili il problema "della fermata" (se l'esecuzione di un dato problema avrà termine partendo da un certo input) e quello del "controllo dei programmi" (sapere se un programma termina e fornisce la soluzione corretta per tutti gli input ammissibili). Fra i decidibili, poi, per molti "non siamo ancora riusciti a trovare l'algoritmo più veloce, o a dimostrare che sotto un certo limite non si può andare"; abbiamo dunque problemi "buoni" (a complessità polinomiale) e problemi "cattivi" (esponenziali o superpolinomiali) o *intrattabili* che richiedono quantità di tempo irragionevoli anche a partire da piccoli input. Una vasta classe di problemi decidibili, inoltre, non è noto se sia trattabile, è assai difficile cioè "stabilire se un algoritmo risponde "sì" con un dato input, ma è facile certificare che lo fa, quando lo fa"; l' "orario" di una scuola, note le disponibilità degli insegnanti e le ore di lezione per materia e per classe, è di questo genere. Se non si può far nulla con gli indecidibili, fra i decidibili la ricerca è volta a migliorare gli intrattabili (mediante soluzioni approssimative ma vicine alle esatte) e ad esplorare la classe dei forse trattabili (se esistono algoritmi buoni per uno, saranno buoni per tutti); a tali frontiere: *calcolo parallelo*, *randomizzazione*, *computazione quantistica*, *calcolo molecolare*, Harel dedica uno smilzo capitolo e non eccessive speranze. Più concreta è la parte dedicata alla crittografia, la quale "sfrutta con astuzia e senza vergogna le limitazioni alla computazione che abbiamo visto finora" ed è largamente adoperata nelle relazioni umane e… tra computer (protocolli di interazione), compagni ormai insostituibili ma non migliori di noi: "è il lato splendente dell'informatica, a cui questo libro cerca di contrapporre il lato oscuro".

Pro

Lettura agile e alla portata dell'utente medio o del semplice curioso.

Contro

Non aggiunge quasi nulla ad un precedente lavoro teorico del 1987.

Giuseppe Cornacchia

Campagna abbonamenti 2005

SI!

MI ABBONO ALLE RIVISTE INFOMEDIA

	PREZZO ABBONAMENTO	
Computer Programming (11 numeri)	€ 50,00	☐
DEV (11 numeri)	€ 50,00	☐
Visual Basic Journal (6 numeri)	€ 44,00	☐
Login (6 numeri)	€ 44,00	☐
CP Web (11 numeri)	€ 42,00	☐
DEV Web (11 numeri)	€ 42,00	☐
VBJ Web (6 numeri)	€ 42,00	☐
Login Web (6 numeri)	€ 42,00	☐

		CP	DEV	VBJ	Login
2 riviste cartacee a scelta	€ 82,00	☐	☐	☐	☐
3 riviste cartacee a scelta	€ 107,00	☐	☐	☐	☐
4 riviste cartacee	€ 132,00	☐	☐	☐	☐

		CP Web	DEV Web	VBJ Web	Login Web
2 riviste Web a scelta	€ 75,00	☐	☐	☐	☐
3 riviste Web a scelta	€ 90,00	☐	☐	☐	☐
4 riviste Web	€ 110,00	☐	☐	☐	☐
4 riviste cartacee+ 4 riviste Web	€ 220,00	☐			

Gli abbonamenti decorrono dal primo numero raggiungibile. Per attivazioni retroattive contattaci al numero 0587/736460 o invia un'e-mail all'indirizzo abbonamenti@gruppoinfomedia.it.

I prezzi degli abbonamenti per l'estero sono maggiorati di spese di spedizione.

GRUPPO EDITORIALE INFOMEDIA S.R.L.
Via Valdera P. 116 - 56038 Ponsacco (Pi) - Tel. 0587/736460 - **Fax 0587/732232**
abbonamenti@infomedia.it

Per abb.ti spedizione posta prioritaria aggiungere:

- **+ euro 18,00 per CP**
- **+ euro 18,00 per DEV**
- **+ euro 10,00 per VBJ**
- **+ euro 10,00 per Login**

INDIRIZZO DI SPEDIZIONE

CODICE CLIENTE _____

Nome/Società

Indirizzo

CAP Città Prov.

Telefono

Fax

E-Mail

MODALITA' DI PAGAMENTO

- ☐ Allego fotocopia del Bonifico Bancario effettuato sul **C/C 000010005424** CAB 71120 ABI 6370 On "M" - Cassa di Risparmio di Volterra - Agenzia di Ponsacco
- ☐ Allego fotocopia della ricevuta del versamento sul C/C Postale **N.14291561** intestato a "Gruppo Editoriale Infomedia S.r.l. - Ponsacco"
- ☐ Allego assegno bancario intestato a "Gruppo Editoriale Infomedia S.r.l." NON TRASFERIBILE
- ☐ Contrassegno (+ € 7,00 contributo spese postali)
- ☐ Autorizzo l'addebito dell'importo di € _____ sulla mia CARTASI'/VISA

N. _____ _____ _____ _____

Scad. ___ ___ mm/aa

Nome del Titolare _____

Data di nascita _____

Firma del Titolare _____

- ☐ Collegati al sito www.infomedia.it dove potrai effettuare il pagamento con carta di credito in modalità sicura (banca SELLA)

PRIVACY
Si autorizza al trattamento dei dati personali ai sensi delle vigenti norme sulla privacy

FIRMA _____

L'IVA sul prezzo dell'abbonamento è assolta dall'Editore e non sussiste l'obbligo di emissione della fattura, ai sensi del D.M. 9 aprile 1993, art.1, comma 5; pertanto, ai fini contabili, farà fede la sola ricevuta di pagamento; perciò la fattura verrà emessa solo se esplicitamente richiesta al momento dell'ordine.

Garanzia di riservatezza - Gruppo Editoriale Infomedia garantisce la massima riservatezza dei dati da lei forniti e la possibilità di richiederne gratuitamente la rettifica o la cancellazione scrivendo a: Responsabile Dati - Gruppo Editoriale Infomedia Srl - Via Valdera P. 116 - 56038 Ponsacco (PI). Le informazioni custodite nel nostro archivio elettronico verranno trattate in conformità alla legge 675/96 sulla tutela dati personali.