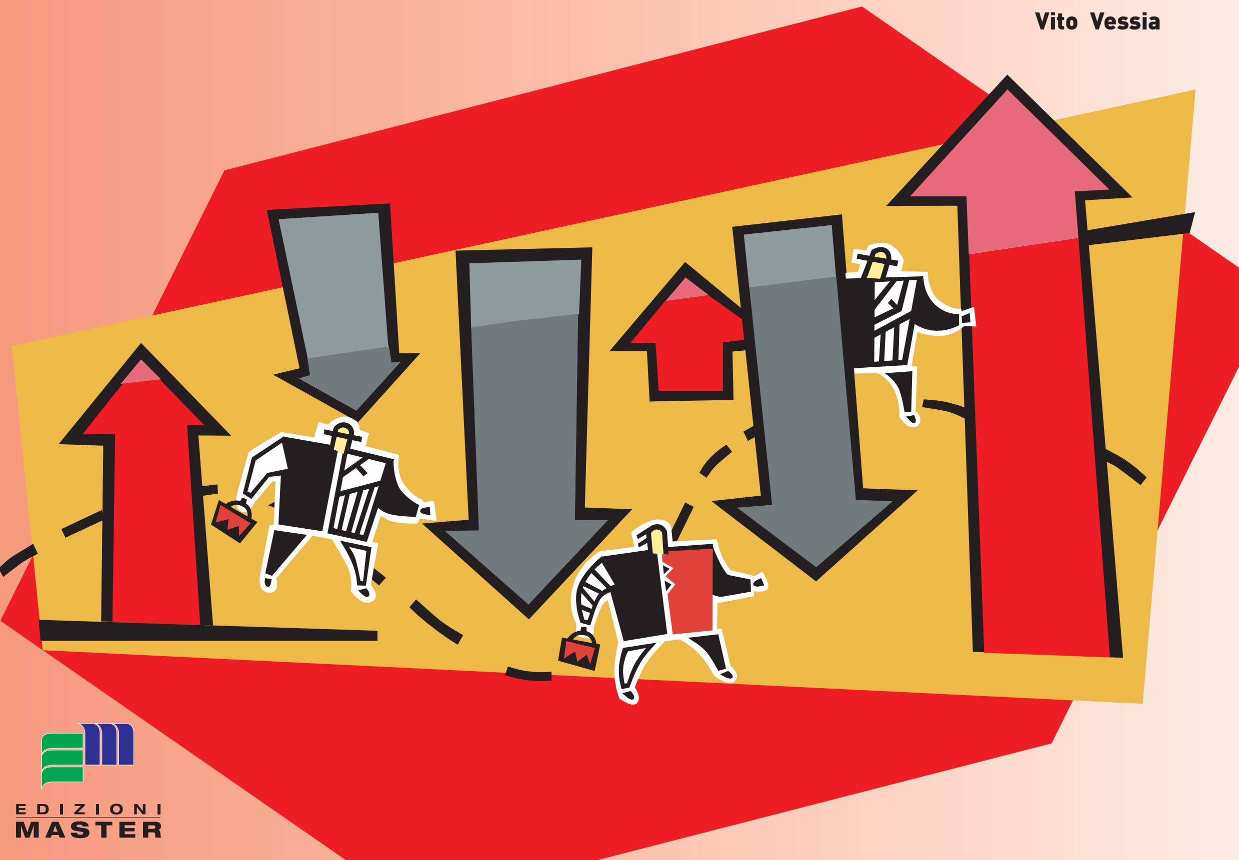
I LIBRI di

ioProgrammo

LA GUIDA DI RIFERIMENTO PER IMPARARE VELOCEMENTE AD UTILIZZARE IL DATABASE DI MICROSOFT

LAVORARE CON SQL SERVER

Vito Vessia



EDIZIONI MASTER

i libri di

ioPROGRAMMO

LAVORARE CON SQL SERVER

Vito Vessia

EDIZIONI MASTER

INDICE

Prefazione 7
Ringraziamenti 8
Introduzione 10
Scopo del volume 11
Requisiti per la corretta fruizione del volume 13

Cenni sul database relazionale

1.1 Cos'è un database relazionale 15
1.2 Breve storia dei database 15
1.3 Anatomia di un database relazionale 16
1.3.1 Tabelle 16
1.3.2 Campi 18
1.3.3 Record 18
1.3.4 Tipi di chiavi 18
2.3.5 Viste 21
1.4 Relazioni 22
1.4.1 Relazione uno a uno 23
1.4.2 Relazione uno a molti 25
1.4.3 Relazione molti a molti 27
1.5 Regole di progettazione del database 29
1.5.1 La scelta dei nomi 30
1.5.2 Prima forma normale 30
1.5.2 Seconda forma normale 31
1.5.2 Terza forma normale 32
1.6 Tipi di dati 34
1.6.1 Tipi di dati numerici 34
1.6.2 Tipi di dati stringa 35
1.6.3 Tipi di dati data e ora 36
1.6.4 Tipi di dati binari 36
1.6.5 Tipi di dati speciali 37

PuntoInformatico | Libri

1.6.6 Tipi di dati definiti dall'utente 38
1.6.7 Valori NULL 38

ISTALLAZIONE E AMMINISTRAZIONE

2.1 Le edizioni di SQL Server 2005 41
2.2 Istallazione del prodotto 44
2.2.1 Le istanze di SQL Server 48
2.2.2 Servizi e modalità di autenticazione 49
2.2.3 Opzioni di collation 51
2.2.4 Completamento dell'istallazione 53
2.3 SQL Server Management Studio 54
2.3.1 Registrazione dei server 57
2.3.2 Avvio e arresto di SQL Server 58
2.4 Query Editor 58
2.5 Amministrazione di SQL Server 64
2.5.1 Configurazione e tuning del database Engine 65
2.5.2 Creazione di un database 67
2.5.3 Schemi, tabelle e campi 71
2.5.4 Relazioni, chiavi esterne, indici e constraint 73
2.5.5 Creazione di viste 78
2.6 Backup e Restore 79
2.6.1 Restore del database da SQL Management Studio . . .82
2.7 Attach e Detach 84

IL LINGUAGGIO TRANSACT-SQL

3.1 Introduzione al linguaggio SQL 87
3.1.1 Storia di SQL 88
3.2 Lo statement select 89
3.2.1 Le clausole obbligatorie Select e From 91
3.2.2 La clausola Where 101
3.2.3 La clausola Group by 107
3.2.4 La clausola Having 112
3.2.5 La clausola Order by 113

3.2.6 La clausola Compute 115
3.3 Ragionare per insiemi 117
3.3.1 La teoria 118
3.4 Gli operatori di insieme 119
3.4.1 Operatori Union e Union all .119
3.4.2 Il nuovo operatore Except 122
3.5 Combinazione di tabelle 122
3.5.1 Cross Join 123
3.5.2 Inner Join 124
3.5.3 Outer Join 129
3.5.4 Subquery 132
3.5.5 CTE (Common Table Expression) 135

PROGRAMMARE SQL SERVER IN T-SQL

4.1 Inserimento di dati nel database 139
4.1.1 Insert 139
4.1.2 Update 141
4.1.3 Delete 142
4.2 Gestione delle transazioni 143
4.2.1 Livelli di isolamento 145
4.3 Creazione e modifica di oggetti del database 146
4.3.1 Creazione, modifica e cancellazione di un database .146
4.3.2 Creazione, modifica e cancellazione di una tabella . .147
4.3.1 Creazione, modifica e cancellazione di una vista 149
4.3.1 Creazione, modifica e cancellazione di un indice 151

PuntoInformatico | Libri

PREFAZIONE

The most exciting phrase to hear in science, the one that heralds the most discoveries, is not "Eureka!", but "That's funny..."

Isaac Asimov

La maggior parte di quanti attualmente si guadagnano da vivere scrivendo software gestionali, categoria a suo modo ascrivibile a quella dei programmatori (almeno lo spero per me!), non è altro che il risultato anagraficamente maturo di un giovine fanciullino che sognava di programmare i videogame e di avviare guerre termonucleari-globali in modalità a caratteri 80 x 25.
Certo, si racconta che qualcuno davvero sia finito a scrivere routine che muovono omini 3D in labirinti, ma si racconta pure che Elvis Presley sia ancora vivo e si diletti a fare l'ultracorpo dentro un marziano piccolo e verde; si tratta, in entrambi i casi, di informazioni non confermate.
Invece noialtri, che siamo personcine ben piantate in terra e troviamo il nostro trastullo nel disegnare basi di dati con almeno trentacinque-trentasei livelli di normalizzazione, nello scrivere sagaci stored procedure, forse anche ricorsive, e nel gestire gli eventi più imprevisti con salaci trigger INSTEAD-OF, non possiamo che sorridere condiscendenti ai nostri giovani colleghi - non ancora avvezzi a cristallizzare le loro fortunate venture in leggi universali - quando questi ci fanno notare che il linguaggio SQL è noioso, che non c'è bisogno di una laurea per scrivere una query e che, quando anche si riesce a farlo tirando fuori un costrutto complicato che inorgoglisce l'autore di cotanta prodezza, non si fa che soddisfare solo il minimo sindacale... Sì sì, ecco che la vena paternalistica che è dentro di noi viene fuori e giù col spiegare a questo giovine improvvido che INNER JOIN è bello, che le viste sono il dono che il Signore Codd ha fatto a noi figli del plain text e che non puoi davvero considerarti un buon informatico se non programmi da almeno tre anni o centomila record...

PuntoInformatico | Libri

Poi magari, finita la giornata, mentre guidiamo alacremente per tornare a casuccia, la nostra mente comincia a fantasticare di argutissime SELECT per tirare fuori il numero di caricatori ancora a disposizione per il Gatling Cannon da usare nel giochino che abbiamo in mente da anni...

RINGRAZIAMENTI

Quando scrissi il mio primo manuale, ormai qualche anno fa, ritenevo la compilazione dei ringraziamenti e, più in generale la scrittura dell'introduzione, tra i momenti fondanti dell'opera compilativa. Poi, col tempo, mi sono ricreduto. Ad esempio, questa volta sono arrivato così distrutto alla meta, per via del grande sforzo da realizzare in poco tempo, che non posso che ringraziare soprattutto l'essere giunto finalmente al termine e dunque ringrazio... il tempo!
In secondo d'ordine, ringrazio Nica e l'allergia al polline che, per ragioni diverse, mi hanno negato il piacere bucolico lasciandomi senza possibilità di distrazione e quindi tutto impegnato all'adempimento del dovere. Ringrazio Edward Codd e Antonio Pinnelli ai quali si deve, per ragioni lievemente diverse, il proselitismo e la diffusione nel mondo (anche quello piccolo) delle databasiche cose. Ringrazio la Eurogest s.r.l., l'azienda per cui lavoro, per aver evitato di chiamare i pompieri ogni volta che mi vedevano arrivare in ritardo senza preavviso e i miei colleghi Biagio e Massimo, del gruppo di sviluppo, che mi permettono di scrivere un report in master/detail ogni volta che sono un po' giù perché sanno che per me ha lo stesso effetto che fare un giro sul Roller Coaster. Ringrazio il mio amico fraterno Andrea Roncone, la cui ontologia, suo malgrado, fa apparire gli affanni di noi poveri informatici solo del pleonastico rumore di fondo. Ringrazio, per il suo incuriosito supporto morale, Annalucia Curci, creatura affascinante e bizzarra come sempre lo sono le equazioni differenziali, la metempsicosi e la Critica di Kant: tutte cose di difficile intelligibilità. Ringrazio le mie amiche e i miei amici più cari e

la mia famiglia che, per ragioni del tutto simili anche se incomprensibili, continuano a credere in me nonostante non abbiano ancora compreso cosa io esattamente faccia, ma dato che la paga mi arriva quasi regolarmente, sono giunti alla conclusione che vale la pena che io continui a fare ciò che faccio, supportandomi con il loro affetto. E ringrazio Nicole, infine, dolce fanciulla che parla l'inglese come Shakespeare e l'italiano come Franco Franchi: she is lonely for me. È l'auspicio.

Buona lettura.

Vito Vessia progetta e sviluppa applicazioni e framework in .NET, COM(+) e Delphi occupandosi degli aspetti architetturali. Prima che il segreto si disveli portando a bizzarri effetti collaterali post-editoriali, i suoi colleghi sanno di lui che con i database non è mai stato amore a prima vista, ma nemmeno a seconda vista. Ma poi si è dovuto ricredere ed è arrivato alla conclusione che sono, sì, noiosi, a volte limitati, immancabilmente cervellotici e indubbiamente non esaltanti, ma sono estremamente utili e permettono di "tirare la carretta". Scrive da anni per le principali riviste italiane di programmazione ed è autore del libro "Programmare il cellulare", Hoepli, 2002, sulla programmazione dei telefoni cellulari connessi al PC con protocollo standard AT+. Può essere contattato tramite e-mail all'indirizzo vvessia@katamail.com e si può consultare il suo sito http://www.codeBehind.it anche per aggiornamenti e download di materiale relativo al volume.

PuntoInformatico | Libri

INTRODUZIONE

Microsoft SQL Server è il sistema di database relazionale (RDBMS) prodotto e sviluppato da Microsoft. Affonda le sue origini nel prodotto Sybase SQL Server 3 per Unix, da cui però si è fortemente evoluto e differenziato a partire dalla versione 7, che di fatto rappresenta una vera e propria riscrittura del code base. L'obiettivo originario, agli inizi degli anni 90, era quello di sviluppare una versione del prodotto per il sistema operativo OS/2. Tuttavia questa decisione è stata quasi da subito disattesa, seguendo la più generale strategia di uscita adottata da Microsoft nei confronti di OS/2, a favore del suo Windows NT. Nel 1994 la collaborazione con Sybase non viene rinnovata e da allora SQL Server subisce un fork di codice notevole. SQL Server 2005 è il nome commerciale dell'ultima versione del prodotto RDBMS di Microsoft e durante tutta la sua fase di sviluppo ha mantenuto il nome in codice Yukon (un'area geografica del Canada). Come tutti i prodotti Microsoft, SQL Server 2005 gira solo su piattaforma Windows e in particolare su Windows 2000 (Desktop e Server), Windows XP e Windows Server 2003. Questo consente al prodotto di trarre il massimo dalla piattaforma S.O. su cui si basa, permettendo una politica di fortissima integrazione (dal file system, alla gestione della sicurezza e all'uso ottimizzato delle API di sistema), proprio perché non ha velleità di prodotto multipiattaforma. Inoltre gli consente di mantenere il look & feel dei tipici prodotti per Windows oltre ad offrire una facilità di installazione che non ha pari nelle versioni per Windows di altri prodotti multipiattaforma. Ma probabilmente il maggior vantaggio rispetto ad altri prodotto concepiti su piattaforma Unix (praticamente tutti gli altri) è la scalabilità: ci sono versioni di SQL Server 2005, come la Express, che girano su portatili non particolarmente dotati e, dall'altra parte, ci sono versioni come la Enterprise che girano su potenti server in cluster. Con Windows come unico collante.

Scopo del volume

Questo manuale fornisce una panoramica completa delle funzionalità e delle caratteristiche di SQL Server 2005. Per ragioni di spazio non si propone come una guida completa ed estensiva, ma vuole fornire gli elementi essenziali per far conoscere questo potente prodotto agli sviluppatori che non si siano ancora avvicinati ad esso e permettere agli altri di approfondirne alcune conoscenze. Si compone di due volumi di cui questo ne costituisce il primo. I due volumi hanno una continuità logica, ma possono essere fruiti anche separatamente perché si focalizzano su aspetti differenti dell'uso del prodotto. Il primo volume si occupa degli aspetti teorici dei database relazionali, del linguaggio T-SQL e dell'installazione, configurazione e tuning di SQL Server 2005. Il secondo volume, invece, è focalizzato sulla programmazione di e con SQL Server 2005, con il linguaggio T-SQL e con il nuovo supporto a Microsoft .NET e degli aspetti avanzati introdotti dalla nuova versione del prodotto, sempre nell'ottica dello sviluppo del software.

PuntoInformatico | Libri

Il Volume 1

È essenzialmente un volume orientato ai progettisti di database. Il volume, però, non vuole essere indirizzato solo a coloro che già sono esperti di SQL Server e delle problematiche dei database relazionali, ma fornisce un'introduzione alla filosofica degli RDBMS e al linguaggio SQL per coloro che si avvicinano per la prima volta a questo particolare mondo dell'informatica e della programmazione.
Il primo capitolo affronta un'introduzione ampia ai concetti, alla filosofia e alle regole su cui si basano i database relazionali, spiegando concetti elementari come tabelle, righe, colonne e tipi di dati, per arrivare poi ad argomenti meno immediati come le regole di buona costruzione e normalizzazione dei database.
Il secondo capitolo rappresenta la sezione di amministrazione del volume: parte dai requisiti per l'installazione di SQL Server e dalla sua in-

stallazione guidata, passo per passo, per poi introdurre SQL Server Management Studio, lo strumento integrato di amministrazione, progettazione e gestione di SQL Server 2005. Dopo una rapida panoramica strumento, vengono affrontati i tipici aspetti sistemistici di gestione: dalla creazione di un database, delle tabelle, delle relazioni tra esse, alle le viste, le stored procedure, le funzioni e i trigger. Per poi passare alla gestione dei backup e dei restore facendo cenno alle novità introdotte da SQL Server 2005, all'attach e al detach al volo di database.
Il terzo capitolo è integralmente dedicato al linguaggio SQL e al suo dialetto Transact SQL adoperato all'interno di SQL Server. Si tratta di un lungo capitolo dedicato quasi esclusivamente all'istruzione SELECT, il cuore del linguaggio SQL e in generale il motore fondamentale che rende la fruizione dei database agevole anche ai comuni mortali.
Il volume non ha evidentemente ambizioni di piena completezza perché, nonostante gli sforzi profusi in tal senso, questo resta sempre un volumetto e SQL Server 2005 resta sempre, dal canto suo, un prodotto complesso e ricco di numerose sfaccettature. Dunque le omissioni non mancheranno, ma questo deve essere considerato uno primo approccio per coloro che si avvicinano all'argomento e, per tutti gli altri, una fonte da cui attingere solo alcuni argomenti di approfondimento.
Nella speranza che il lavoro sia di gradimento del lettore, l'appuntamento è al prossimo numero con il Volume 2 che completa l'opera.

Requisiti per la corretta fruizione del volume

Questa guida fa riferimento alla versione standard di SQL Server 2005, dotata di versione completa del tool SQL Management Studio. Però la gran parte delle informazioni è applicabile anche alla versione Express, distribuita gratuitamente da Microsoft insieme ad una

versione leggera dello strumento di amministrazione, SQL Mangement Studio Express. Pertanto, coloro che si avvicinano per la prima volta a questo prodotto o per gli altri utilizzatori di SQL Server che non hanno ancora fatto la migrazione alla versione 2005, la versione gratuita Express offre un'ottima possibilità di studiare e scoprire il prodotto a costo zero. Peraltro, la licenza che accompagna la versione Express la rende di fatto un prodotto vero e non solo una versione vetrina-giocattolo perché con la Express è possibile sviluppare vendere applicazioni che si basano su questa versione. Senza costi per lo sviluppatore e per l'utente finale, garantendo così un percorso di migrazione molto scalabile verso le versioni successive a pagamento e rappresentando una valida alternativa a prodotti come Microsoft Access o a soluzioni più avanzate open source come MySQL o Firebird.
SQL Server 2005 Express è scaricabile gratuitamente dal seguente indirizzzo:
http://www.microsoft.com/downloads/details.aspx?FamilyID=220549B5-0B07-4448-8848-DCC397514B41&displaylang=it
La versione leggera dello strumento di amministrazione, SQL Management Studio Express, è disponibile in download gratuito dal seguente indirizzo:
http://www.microsoft.com/downloads/details.aspx?familyid=C243A5AE-4BD1-4E3D-94B8-5A0F62BF7796&displaylang=it
Entrambi prodotti sono disponibili in lingua italiana.
Nel volume si è fatto riferimento a tre database di esempio: AdventureWorks, il vero database di riferimento della versione 2005, Northwind, il database che ha accompagnato migliaia di programmatori nello studio SQL Server 7.0 e SQL Server 2000 e l'antico e un po' superato anche se sempre valido pubs.
Il primo è fornito in dotazione con SQL Server 2005 nelle versioni a pagamento. Gli utilizzatori della versione Express possono scaricare lo script T-SQL di creazione di AdventureWorks da questo indirizzo:

PuntoInformatico Libri

http://www.microsoft.com/downloads/details.aspx?FamilyID=e719ecf7-9f46-4312-af89-6ad8702e4e6e&DisplayLang=en

Northwind e Pubs sono invece disponibili per tutti (infatti non vengono distribuiti più con SQL Server 2005, nemmeno nelle versioni commerciali), al seguente indirizzo:

http://www.microsoft.com/downloads/details.aspx?familyid=06616212-0356-46a0-8da2-eebc53a68034&displaylang=en

Se vi risulta troppo complicato riportare questi indirizzi, usata il buon vecchio Google usando le keyword "SQL Server 2005 Samples"... Una volta scaricati i setup ed installati, è sufficiente aprire i file .Sql da SQL Management Studio (o Express) ed eseguirli. Genereranno l'intero database compresi i dati di esempio preziosi per i nostri esempi e per il vostro studio.

CENNI SUL DATABASE RELAZIONALE

Questo è un volume sul motore di database Microsoft SQL Server ed in particolare sulla versione 2005 e sulle numerose novità introdotte. Ma Microsoft SQL Server 2005, che per brevità da ora chiamerò semplicemente SQL Server, prima di essere un potente concentrato di novità e di tecnologia, è innanzitutto un database relazione. Ecco il perché di questa breve introduzione,

1.1 COS'È UN DATABASE RELAZIONALE

Un database è un insieme organizzato di dati, utilizzato per modellare gerarchie, processi organizzativi, modelli organizzativi e, più in generali, per rappresentare modelli logici in cui salvare e da cui recuperare informazioni. Si tratta, dunque, di un concetto logico e non del medium fisico che ospiterà i dati; pertanto non fa differenza se verrà conservato sulla carta o all'interno di un computer: se ha certe caratteristiche, un sistema di questo tipo verrà comunque assimilato al concetto di database.
L'aggettivo relazionale qualifica una caratteristica aggiuntiva di un database indicandone un'appartenenza ben precisa e distinguendolo da altri modelli. Infatti, il modello di modellazione e di salvataggio dei dati che usiamo tutti i giorni prevalentemente e che è ben rappresentato da SQL Server, non è l'unico modello di database possibile, ma è solo il più diffuso ed universalmente accettato.
Esistono database gerarchici, database reticolari, database ad oggetti ecc... Ma questa è un'altra storia...

1.2 BREVE STORIA DEI DATABASE

Le origini del modello Relazionale sono un passaggio assolutamente affascinante ed immancabile per ogni volume che, anche lontanamente, tratti di problematiche sui database. Esse sono da far risalire al lontano 1969, quando Edgard F. Codd, allora ricercatore della

PuntoInformatico | Libri

IBM, cercava di modellare logicamente nuove metodologie per gestire grandi masse di dati. Era un matematico di professione, pertanto cercò di adattare alcuni concetti matematici per applicarli al problema che intendeva risolvere e così fece, rendendo di fatto la Teoria dei Database Relazionali, una sorta di branca della Matematica applicata. Il suo studio sfociò, l'anno successivo, con la pubblicazione di quella che viene considerata la Sacra Bibbia dei progettisti, degli amministratori, dei programmatori o semplicemente degli amanti della materia: "Un Modello Relazionale di dati per Grandi Banche dati condivise". Il volume introdusse tutti i concetti ancora oggi alla base del modello relazionale, che si basa su due teorie matematiche: la Teoria degli Insiemi e la Logica dei Predicati di Primo Grado; entrambe queste teorie possono tranquillamente essere ignorate e nonostante questo è possibile essere, al contempo, un grande esperto di database del modello relazionale. Lo stesso termine "relazione" deriva dalla teoria degli insiemi e non dalle relazioni tra le tabelle, come comunemente si crede. L'acronimo più comunemente usato per identificare i database è R.D.B.M.S. (solitamente indicato senza i puntini), che sta proprio per Rational DataBase Management System. Le prime implementazioni commerciali del modello Codd apparvero negli anni Settanta e da allora il settore è in continua ed inarrestabile espansione.

1.3 ANATOMIA DI UN DATABASE RELAZIONALE

Nel modello relazionale i dati sono immagazzinati in relazioni che appaiono in forma di tabelle. Ogni relazione è composta da tuple (righe o record) e attributi (campi o colonne).

1.3.1 Tabelle

Le tabelle sono l'elemento strutturale essenziale di un database relazionale. L'ordine logico dei campi e dei record in una tabella non ha alcuna importanza, salvo particolari eccezioni che, nello specifi-

co di SQL Server 2005, attengono agli indici clustered, che però saranno oggetto di una successiva trattazione.
Ogni tabella contiene sempre uno o più campi che ne costituiscono la Chiave Primaria; in altre parole, il valore del campo o l'insieme dei valori della tupla che ne costituisce la Chiave Primaria (o Primary Key) identificano sempre in modo univoco una riga all'interno della tabella. La Figura 1.1 propone un esempio di tabella, Person.Address del database di esempio AdventureWorks di SQL Server 2005 e la colonna AddressId ne costituisce la chiave primaria.

COLONNE (CAMPI)

AddressID	AddressLine1	AddressLine2	City	StateProvinceID	PostalCode	rowguid
1	1970 Napa Ct.	NULL	Bothell	79	98011	9aadcb0d-36cf-...
2	9833 Mt. Dias Blv.	NULL	Bothell	79	98011	32a54b9e-e034-...
3	7484 Roundtree...	NULL	Bothell	79	98011	4c506923-6d1b-...
4	9539 Glenside Dr	NULL	Bothell	79	98011	e5946c78-4bcc-...
5	1226 Shoe St.	NULL	Bothell	79	98011	fbaff937-4a97-4...
6	1399 Firestone ...	NULL	Bothell	79	98011	febf8191-9804-...
7	5672 Hale Dr.	NULL	Bothell	79	98011	0175a174-6c34-...
8	6387 Scenic Ave...	NULL	Bothell	79	98011	3715e813-4dca-...
9	8713 Yosemite Ct.	NULL	Bothell	79	98011	268af621-76d7-...
10	250 Race Court	NULL	Bothell	79	98011	0b6b739d-8eb6-...
11	1318 Lasalle Street	NULL	Bothell	79	98011	981b3303-aca2-...
12	5415 San Gabrie...	NULL	Bothell	79	98011	1c2c9cfe-ab9f-4...
13	9265 La Paz	NULL	Bothell	79	98011	e0ba2f52-c907-...
14	8157 W. Book	NULL	Bothell	79	98011	a1c658ae-c553-...
15	4912 La Vuelta	NULL	Bothell	79	98011	f397e64a-a9d8-...
16	40 Ellis St.	NULL	Bothell	79	98011	0312b65f-cb60-...
17	6696 Anchor Drive	NULL	Bothell	79	98011	ce9b3b47-9267-...
18	1873 Lion Circle	NULL	Bothell	79	98011	963854f7-e3cb-...
19	3148 Rose Street	NULL	Bothell	79	98011	6b7acb0f-cdbf-4...

RIGHE (RECORD)

Figura 1.1: Un esempio di tabella

Questa caratteristica è fondamentale per la sua ripercussione funzionale: l'accesso alle informazioni di una riga in una tabella è indipendente dal suo formato e dalla sua posizione fisica nella tabella stessa perché viene semplicemente contraddistinta dal valore della sua chiave primaria, che è l'unico elemento necessario e sufficiente per identificare e recuperare le informazioni della riga.
Una tabella contiene informazioni coerenti, ad esempio l'anagrafica dei clienti, l'elenco delle strade, l'elenco dei prodotti, l'elenco dei movimenti contabili, la lista degli accessi al programma in una de-

PuntoInformatico Libri

terminata giornata ecc... Dunque è di vitale importanza che il nome della tabella sia quanto più preciso e pertinente possibile e che identifichi con precisioni la natura dei dati contenuti all'interno di essa. E, soprattutto, che dia luogo ad ambiguità di interpretazione.

1.3.2 Campi

Il campo è la struttura più piccola di memorizzazione di un database e costituisce e contiene un valore atomico scalare di una tabella. Dunque, se consideriamo una tabella come una matrice costituita in orizzontale da colonne e in verticale da righe, il campo è precisamente la singola cella e dunque contiene un singolo valore. Un campo contiene un'informazione precisa all'interno della tabella, ad esempio Cognome, Numero di telefono, Prezzo del prodotto ecc... Quindi è importante, come descritto in precedenza per la tabella, che i nomi dei campi sia precisi, pertinenti e che non diano luogo ad ambiguità.

1.3.3 Record

Un record rappresenta la singola istanza del dato in tabella. Esso è composto da una serie di campi definiti secondo la struttura della tabella stessa e tra essi vi è il campo o la tupla di campi della chiave primaria dai cui valori è possibile identificare univocamente il record all'interno della tabella, indipendentemente dalla sua posizione fisica.

1.3.4 Tipi di Chiavi

Le chiavi sono campi speciali che giocano ruoli specifici all'interno della tabella. Ci sono numerosi tipi di chiavi. La chiave primaria (Primary Key o PK) è un campo o un gruppo di campi che identifica in modo univoco ogni record all'interno della tabella, come già indicato in precedenza.
Individuare una chiave primaria all'interno di una tabella non è un'operazione banale, come apparentemente potrebbe sembrare. Os-

Column Name	Data Type	Allow Nulls
EmployeeID	int	☐
NationalIDNumber	nvarchar(15)	☐
ContactID	int	☐
LoginID	nvarchar(256)	☐
ManagerID	int	☑
Title	nvarchar(50)	☐
BirthDate	datetime	☐
MaritalStatus	nchar(1)	☐
Gender	nchar(1)	☐
HireDate	datetime	☐
SalariedFlag	Flag:bit	☐
VacationHours	smallint	☐
SickLeaveHours	smallint	☐
CurrentFlag	Flag:bit	☐
rowguid	uniqueidentifier	☐
ModifiedDate	datetime	☐
		☐

Column Properties

Description	Primary key for Employee records.
Deterministic	Yes
DTS-published	No
Full-text Specification	No
Has Non-SQL Server Subscriber	No
Identity Specification	Yes
(Is Identity)	Yes
Identity Increment	1
Identity Seed	1
Indexable	Yes

Figura 1.2: Chiavi primarie, chiavi candidate e chiavi surrogate

serviamo in Figura 1.2: il diagramma riportato è tratto sempre dal database di esempio AdventureWorks di SQL Server 2005. La tabelle Employee rappresenta l'elenco dei dipendenti di un'azienda. In essa è possibile due potenziali chiavi primarie alternative tra loro: ContactID (un identificativo di contatto del dipendente e quindi certamente univoco) e NationalIDNumber (il numero di previdenza sociale del dipendente, anch'esso di certo univoco. Entrambe queste chiavi sarebbe candidate a divenire chiave primaria della tabella. Infatti per esse è stato introdotto opportunamente il concetto di Chiavi Candidate. Come risolvere questa ambiguità accertandosi al contempo di mantenere l'unicità della chiave? Un espediente consente di superare questo problema: introdurre una Chiave Surrogata, ovvero una chiave artificialmente introdotta che assicuri l'unicità dei valori contenuti nel campo ad essa preposto. La prassi più comune è quello di utilizzare un valore numerico univoco, magari generato in automatico da un contatore. SQL Server dispone a tal proposito del

campo di tipo Identity (Identità) che è un campo numerico sul quale viene attivato un contatore gestito ed incrementato automaticamente dal sistema. Ad ogni nuovo inserimento di record, SQL Server genererà un nuovo valore numerico univoco da inserire nel campo e aggiornerà il contatore corrispondente.

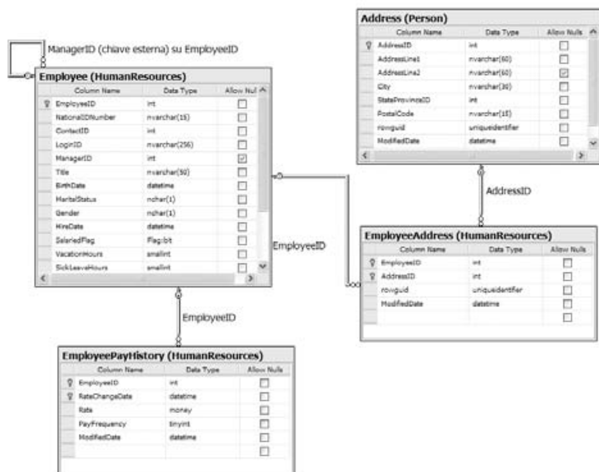


Figura 1.3: Relazioni tra tabelle

È inoltre possibile stabilire relazioni tra diverse tabelle del database. In tal caso, i campi che sostituiscono la chiave primaria di una tabella divengono anche campi della tabella ad essere relazionata. Questo nuovo gruppo di campi nella tabella relazionata diventano la Foreign Key, la chiave esterna, della tabella stessa verso la tabella in relazione.

Il diagramma in Figura 1.3 mostra quattro tabelle in relazione tra loro: Employee (la tabella dei dipendenti), Address (la tabella degli indirizzi), EmployeeAddress (la tabella di relazione tra Employee e Address) e EmployeePayHistory (la tabella delle buste paga dei dipen-

denti). In questo esempio possiamo ritrovare diverse tipologie di relazioni: la relazione diretta tra Employee e EmployeePayHistory, attraverso il campo EmployeeID (ogni riga di EmployeePayHistory contiene un campo Foreign Key verso Employee), una relazione indiretta tra Employee e Address che passa attraverso la tabella di relazione EmployeeAddress e una relazione su se stessa della tabella Employee che è una particolare variante che consente ad una tabella di dotarsi di campi foreign key verso primary key della stessa tabella in modo realizzare relazioni gerarchiche o parentali tra record della stessa tabella. Nello specifico, la tabella Employee, che ha come chiave primaria EmployeeID, è dotata anche della foreign key ManagerID, in relazione su stessa sul campo EmployeeID. La relazione indica il manager responsabile del dipendente che, essendo un dipendente a sua volta, è comunque ospitato dalla tabella Employee.
Le foreign key sono importanti non solo perché aiutano a stabilire legami tra diverse tabelle, ma anche perché aiutano ad assicurare l'integrità referenziale. Questo significa che le registrazioni in entrambe le tabelle saranno sempre legate in modo opportuno e corretto perché i valori delle foreign key dovranno sempre correttamente derivare dai valori delle chiavi primarie della tabella a cui la relazione fa riferimento. In tal modo si evitano record orfani. Questo è uno degli enormi vantaggi del database e si ripercuote anche nello sviluppo di applicazioni che si basano sui database: infatti non si rende più necessario, da parte del programma e del programmatore, il controllo di integrità dei dati perché questo viene gestito, in ultima istanza, direttamente dal database stesso. Nessun programma al mondo, nessun programmatore al mondo e nessun utente del programma al mondo potranno, infatti, associare ad un dipendente un manager che già non esista nell'elenco dei dipendenti.

1.3.5 Viste

Una vista è una tabella virtuale composta da campi appartenenti ad una o più tabelle del database. Tali tabelle di provenienza sono

PuntoInformatico Libri

dette tabelle base. La vista è un oggetto virtuale perché i dati in essa contenuti non risiedono fisicamente al suo interno, ma direttamente nelle tabelle base che la costituiscono creando, dunque, una vera e propria vista alternativa su quei dati. L'unica informazione effettivamente conservata nella vista è la sua struttura e cioè l'elenco dei campi che la costituisce e il riferimento alle tabelle base e ai relativi campi di origine.
Le viste in SQL Server vengono sempre create a partire da query SQL di selezione. SQL Server 2005 offre il supporto alle viste modificabili e cioè a viste nelle quali è possibile anche scrivere dati e non solo recuperarli. Data la natura intrinseca della vista, l'eventuale operazione di scrittura si trasforma, poi, in accessi fisici in scrittura alle tabelle base della vista. In Figura 1.4 è possibile osservare il diagramma schematico dell'intreccio di tabelle base della vista vEmployee del nostro solito database AdventureWorks di riferimento.

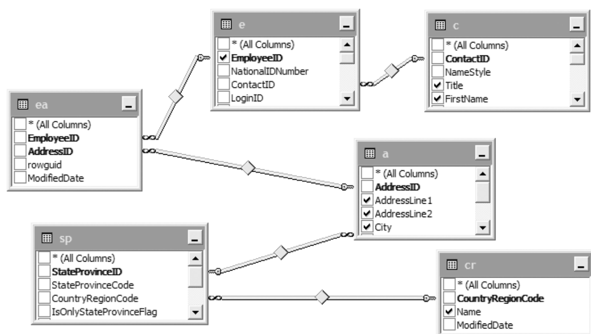


Figura 1.4: Una esempio in “vista”

1.4 RELAZIONI

Mettere in relazione un elemento è uno degli aspetti più interessanti e rappresentativi di un database relazionale. Ciò che rende interessante un database relazionale è la capacità di proiettare le relazio-

ni e gli intrecci delle tabelle al di fuori del modello fisico sottostante e senza esserne condizionati. Il vero scopo delle relazioni, infatti, è quello di riprodurre un modello di realtà e i legami in esso contenuti usando gli strumenti e le teorie messe a disposizione da SQL Server e non, invece, il contrario; cioè farsi condizionare dal modello fisico per adattare il modello reale ad esso.
Esistono diversi modelli di relazione. Osserviamoli in dettaglio.

1.4.1 Relazione Uno a Uno

Due tabelle si dicono in relazione Uno a Uno quando ad ogni riga di una corrisponde una sola riga dell'altra e viceversa. Per questo esempio useremo un altro famoso database di esempio di SQL Server, il buon vecchio Pubs. In Figura 1.5 è possibile osservare la tabella Publishers che rappresenta una lista di editori. Ciascun editore è identificato da una chiave primaria, pub_id, da una denominazione, la città di ubicazione, lo stato (o la provincia) di appartenenza e la nazione. Ad estensione di queste informazioni è stata prevista una seconda tabella, pub_info, che qualifica, per ciascun editore, anche un logo e una descrizione aggiuntiva (pr_info).

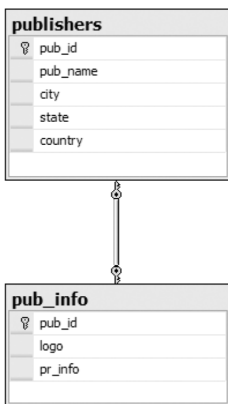


Figura 1.5: Una relazione Uno a Uno

Naturalmente la chiave primaria di questa seconda tabella è sempre pub_id proprio ad indicare l'assoluta corrispondenza biunivoca (uno a uno) tra le due tabelle. Infatti ogni riga di questa seconda tabella contiene i dati necessari ad estendere le informazioni della prima. Tale collegamento avviene proprio attraverso il campo pub_id: agganciando le due tabelle tramite questo campo si può notare come ad una riga della prima corrisponda una ed una sola riga della seconda.

publishers

	pub_id	pub_name	city	state	country
▶	0736	New Moon Books	Boston	MA	USA
	0877	Binnet & Hardley	Washington	DC	USA
	1389	Algodata Infosy...	Berkeley	CA	USA
	1622	Five Lakes Publis...	Chicago	IL	USA
	1756	Ramona Publishers	Dallas	TX	USA
	9901	GGG&G	M□nchen	NULL	Germany
	9952	Scootney Books	New York	NY	USA
	9999	Lucerne Publishing	Paris	NULL	France
*	NULL	NULL	NULL	NULL	NULL

pub_info

	pub_id	logo	pr_info
▶	0736	<Binary data>	This is sample text data for New Moon Books, publisher 0736 i
	0877	<Binary data>	This is sample text data for Binnet & Hardley, publisher 0877 i
	1389	<Binary data>	This is sample text data for Algodata Infosystems, publisher 1
	1622	<Binary data>	This is sample text data for Five Lakes Publishing, publisher 16
	1756	<Binary data>	This is sample text data for Ramona Publishers, publisher 175
	9901	<Binary data>	This is sample text data for GGG&G, publisher 9901 in the pub
	9952	<Binary data>	This is sample text data for Scootney Books, publisher 9952 in
	9999	<Binary data>	This is sample text data for Lucerne Publishing, publisher 999
*	NULL	NULL	NULL

Figura 1.6: Contenuto delle tabelle della relazione Uno a Uno a confronto

Questo tipo di relazione non è molto frequente all'interno dei database, soprattutto per motivi prestazionali: infatti, per accedere ad un'informazione che è logicamente unica ed atomica, bisogna fare due accessi fisici a tabelle distinte. Pertanto si tende a preferire di inserire tutto all'interno di un'unica tabella.

1.4.2 Relazione Uno a Molti

Questo modello di relazione è più comune: un singolo record della prima tabella può essere legato a molti record della seconda tabella, ma un singolo record della seconda tabella può essere legato con un unico record della prima tabella.

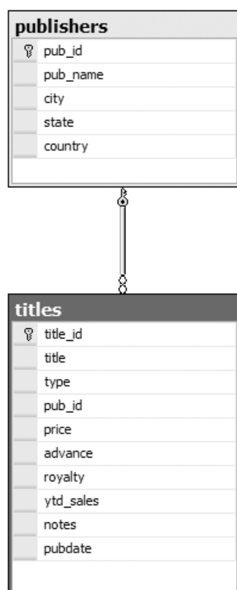


Figura 1.7: Una relazione Uno a Molti

PuntoInformatico Libri

Usando il database Pubs come riferimento, potremmo considerare la relazione tra la tabella Titles (l'elenco dei titoli dei volumi pubblicati) e la tabella Publishers (gli editori). Ciascun volume è identificato dalla chiave primaria title_id ed è evidentemente pubblicato da un editore identificato dalla foreign key pub_id che si relazione proprio con la tabella Publishers (chiave primaria pub_id).

publishers

	pub_id	pub_name	city	state	country
▸	[illegible]	New Moon Books	Boston	MA	USA
	0877	Binnet & Hardley	Washington	DC	USA
	1389	Algodata Infosy...	Berkeley	CA	USA
	1622	Five Lakes Publis...	Chicago	IL	USA
	1756	Ramona Publishers	Dallas	TX	USA
	9901	GGG&G	M□nchen	NULL	Germany
	9952	Scootney Books	New York	NY	USA
	9999	Lucerne Publishing	Paris	NULL	France
*	NULL	NULL	NULL	NULL	NULL

titles

	title_id	title	type	pub_id	price	advance	royalty	ytd_sales	notes
1	BU2075	You Can Combat Computer Stress!	business	0736	2,99	10125,00	24	18722	The late
2	PS2091	Is Anger the Enemy?	psychology	0736	10,95	2275,00	12	2045	Carefull
3	PS2106	Life Without Fear	psychology	0736	7,00	6000,00	10	111	New ex
4	PS3333	Prolonged Data Deprivation: Four Case Studies	psychology	0736	19,99	2000,00	10	4072	What h
5	PS7777	Emotional Security: A New Algorithm	psychology	0736	7,99	4000,00	10	3336	Protecti
6	TC3218	Onions, Leeks, and Garlic: Cooking Secrets of the M...	trad_cook	0877	20,95	7000,00	10	375	Profuse
7	TC4203	Fifty Years in Buckingham Palace Kitchens	trad_cook	0877	11,95	4000,00	14	15096	More an
8	TC7777	Sushi, Anyone?	trad_cook	0877	14,99	8000,00	10	4095	Detaile
9	PS1372	Computer Phobic AND Non-Phobic Individuals: Beha...	psychology	0877	21,59	7000,00	10	375	A must
10	MC2222	Silicon Valley Gastronomic Treats	mod_cook	0877	19,99	0,00	12	2032	Favorite
11	MC3021	The Gourmet Microwave	mod_cook	0877	2,99	15000,00	24	22246	Traditio
12	MC3026	The Psychology of Computer Cooking	UNDECIDED	0877	NULL	NULL	NULL	NULL	NULL
13	PC1035	But Is It User Friendly?	popular_comp	1389	22,95	7000,00	16	8780	A surve
14	PC8888	Secrets of Silicon Valley	popular_comp	1389	20,00	8000,00	10	4095	Muckra
15	PC9999	Net Etiquette	popular_comp	1389	NULL	NULL	NULL	NULL	A must-
16	BU7832	Straight Talk About Computers	business	1389	19,99	5000,00	10	4095	Annotat
17	BU1032	The Busy Executive's Database Guide	business	1389	19,99	5000,00	10	4095	An over
18	BU1111	Cooking with Computers: Surreptitious Balance Sheets	business	1389	11,95	5000,00	10	3876	Helpful

Figura 1.8: Una relazione Uno a Molti: i dati a confronto

Questo tipo di relazione è molto importante perché introduce un concetto fondamentale nella teoria dei database: la normalizzazione. Infatti, le informazioni relative a ciascun editore sarebbero potuto essere inserite direttamente in campi aggiuntivi della tabelle Titles (ad esempio pub_name, city, state ecc...). Però, come si evince chiaramente dalla Figura 1.8, ci sono diversi volumi pubblicati dallo stesso editore e dunque questo approccio avrebbe irrimediabilmente portato a ridondare i dati riproponendo, ad esempio, cinque volte i dati relativi allo stesso editore identificato dal codice 0736 (New Moon Books) e ben rispettivamente sette volte e sei volte i dati relativi agli editori 0877 (Binnet & Hardley) e 1389 (Algodata Infosystem). Invece, l'aver distribuito queste informazioni su due tabelle, ci ha consentito di mantenere i dati molto più compatti ed evitare così le ripetizioni.

1.4.3 Relazioni Molti a Molti

Esiste un'ulteriore possibilità di relazione, definita Molti a Molti, nella quale ad ognuna delle righe di una tabella possono corrispondere molte righe di un'altra tabella e viceversa. Si tratta di una situazione in cui vi sono molteplici rimandi che sono possibili da gestire soltanto introducendo una terza tabella di appoggio, detta di relazione. In Figura 1.8 ne possiamo osservare un esempio tratto dal solito Pubs: la tabella Titles rappresenta l'elenco dei volumi pubblicati, invece Authors è l'elenco di autori di volumi. Uno stesso autore può scrivere più volumi così come lo stesso volume può essere redatto da più autori. Questo è un classico esempio di relazione molti a molti. Se avessimo utilizzato una sola tabella, avremmo dovuto inserire tutti i campi relativi al volume e tutti i campi relativi all'autore. Inoltre, se il volume fosse stato scritto da più autori, avremmo dovuto scrivere una nuova riga per ciascuno degli autori ripetendo inutilmente, però, anche le informazioni relative al volume. Inoltre, se uno stesso autore avesse scritto più libri, le sue informazioni anagrafiche sarebbero state ripetute su tutte le righe relative ai suoi volumi pubblicati, producendo così un'inutile e ridondante duplicazione di informazioni.

PuntoInformatico Libri

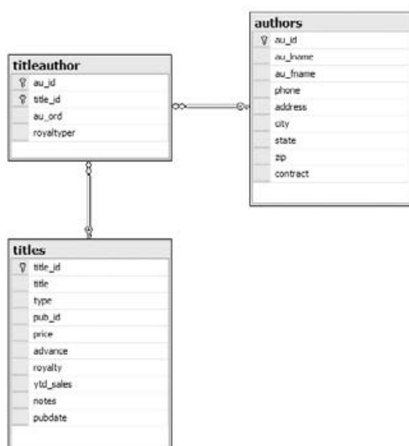
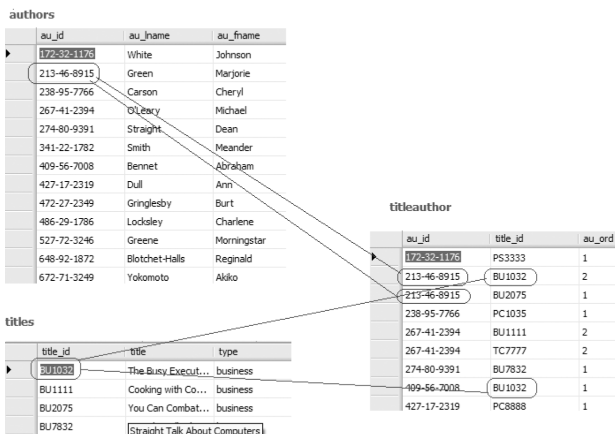


Figura 1.9: Una relazione Molti a Molti

Ma come realizzare una relazione di questo tipo? Il modello Uno a Molti in questo caso è inadeguato perché non impone che una delle due tabelle abbia una sola righe che punta ad molteplici righe dell'altra tabella. Dunque ci viene in aiuto la tabella di relazione. Essa è costituita da una chiave primaria composta da entrambe le chiavi primarie delle due tabelle, e quindi dal campo au_id in foreign key verso Authors e title_id in foreign key verso Titles. E così, mentre Authors e Titles conterranno rispettivamente l'elenco senza ripetizioni di autori e l'elenco senza ripetizioni di volumi, in Titleauthors ci saranno i vari collegamenti.



authors

au_id	au_lname	au_fname
172-32-1176	White	Johnson
213-46-8915	Green	Marjorie
238-95-7766	Carson	Cheryl
267-41-2394	O'Leary	Michael
274-80-9391	Straight	Dean
341-22-1782	Smith	Meander
409-56-7008	Bennet	Abraham
427-17-2319	Dull	Ann
472-27-2349	Gringlesby	Burt
486-29-1786	Locksley	Charlene
527-72-3246	Greene	Morningstar
648-92-1872	Blotchet-Halls	Reginald
672-71-3249	Yokomoto	Akiko

titleauthor

au_id	title_id	au_ord
172-32-1176	PS3333	1
213-46-8915	BU1032	2
213-46-8915	BU2075	1
238-95-7766	PC1035	1
267-41-2394	BU1111	2
267-41-2394	TC7777	2
274-80-9391	BU7832	1
409-56-7008	BU1032	1
427-17-2319	PC8888	1

titles

title_id	title	type
BU1032	The Busy Execut...	business
BU1111	Cooking with Co...	business
BU2075	You Can Combat...	business
BU7832	Straight Talk About Computers	

Figura 1.10: Una relazione Molti a Molti: un esempio

E così, nel caso del volume "The Busy Executive's Database Guide" (codice BU1032), scritto dagli autori 213-46-8915 (Johnson White) e 409-56-7008 (Marjorie Green), in TitleAuthors ci saranno due record rispettivamente con chiave primaria (BU1032, 213-46-8915) e (BU1032, 409-56-7008), esprimendo il caso di un volume scritto da più autori (evidenziato col tratto rosso in Figura 1.9). Invece, l'e-

sempio di un autore che ha scritto più volumi (evidenziato col tratto blu in Figura 1.9) è identificato dall'autrice 213-46-8915 (Marjorie Green) che ha scritto sia il volume BU1032 dell'esempio precedente che il volume BU2075 ("You Can Combat Computer Stress!"). Ed ecco che la nostra brava Marjorie si trova ad aver scritto più volumi (relazione uno a molti tra autore e volumi) e ad aver partecipato ad un volume con altri autori (relazione uno a molti tra volume e autori). Ecco quindi una relazione Molti a Molti. Sebbene più complesso e oneroso, questo modello di relazione è il più potente perché è in grado di rappresentare anche relazioni degli altri due tipi. Infatti nessuno ci vieta di rappresentare con questo metodo anche autori che hanno scritto un solo libro in vita loro e senza altri coautori (relazione uno a uno tra libro ed autore) o autori che hanno scritto più libri ma mai con coautori (relazione uno a molti tra autore e libri).

1.5 REGOLE DI PROGETTAZIONE DEL DATABASE

Progettare un database, strutturare i dati in differenti tabelle, decidere i campi, le chiavi, le viste e le relazioni non è un lavoro meccanico ne esiste una formula automatica universalmente applicabile. Quello che è da tenere sempre ben presente è lo scopo ultimo della progettazione e cioè rappresentare il modello di realtà che dovrà essere contenuto nel database. Solo così sarà possibile, al contempo, ottenere una buona approssimazione nella rappresentazione del modello reale, evitare la ridondanza dei dati ed evitare l'ipercompattazione che, se esasperata, porta una modellazione inefficiente e troppo ostica da gestire nelle query e nelle operazioni di scrittura dei dati. Le relazioni assumono un ruolo fondamentale nella buona progettazione di un database. Il processo di progettazione del database atto alla compattazione dei dati per ridurne la ridondanza è detto normalizzazione ed è stato formalizzato da Codd attraverso una serie di regole dette forme normali. Si tratta di un processo incrementale che, ad ogni stadio successivo, porta sempre ad una maggiore compattazione dei dati.

PuntoInformatico | Libri

1.5.1 La scelta dei nomi

Dare un nome corretto ad una tabella e, ancora di più, ad un campo, riveste un'importanza cruciale. Un nome ambiguo, vago o poco chiaro causerà dei problemi e produrrà incertezze nell'uso dell'informazione contenuta nel campo. È importante che il suo significato sia comunemente accettato da tutto il dominio rappresentato dal database e che non risulti ambiguo perché dotato di significati diversi se letto da persone diverse. A volte può rendersi necessario l'uso di un prefisso per descrivere meglio l'informazione. Se il campo Telefono di una tabella è ambiguo perché non si capisce se è il numero di telefono del cliente o quello del venditore, semplicemente usare un prefisso o un postfisso di specificazione (ad esempio TelefonoCliente). Evitare l'uso di acronimi, se possibile, optando per una specificazione estesa nel significato del nome. Infine, evitare che lo stesso campo possa contenere informazioni diverse e alternative tra loro, soprattutto se di natura eterogenea. Ad esempio al campo CodiceFiscalePartitaIVA si deve preferire lo sdoppiamento in CodiceFiscale e PartitaIVA che, oltretutto, hanno lunghezze diverse (sedici caratteri contro undici) e sono di tipo diverso (alfanumerico l'uno, numerico l'altro).

1.5.2 Prima forma normale

Una tabella di un database si dice non in prima forma normale (1NF) se, e solo se, ciascun attributo può contenere solo valori atomici (ossia non ci sono attributi aggregati o multivalore). Ad esempio, proviamo a pensare cosa succederebbe in termini sia di tempo e di spazio se in una tabella di dati anagrafici di dipendenti comparisse l'Indirizzo del dipendente nella forma "Via Giuseppe De Rosis 23 – 70126 Bari". In Figura 1.11 possiamo osservare un esempio di tabella non normalizzata.

In tal caso il campo contiene evidentemente valori multipli, infatti potremmo suddividerlo nei campi Toponimo, Denominazione della stra-

tabella Soggetti
non normalizzata

Column Name	Data Type	Allow Nulls
IdSoggetto	int	☐
Cognome	varchar(50)	☑
Nome	varchar(50)	☑
CodiceFiscale	varchar(16)	☑
Indirizzo	varchar(255)	☑

IdSoggetto	Cognome	Nome	CodiceFiscale	Indirizzo
1	Vessia	Vito	VSSVTI74M30A6...	Via Giuseppe De Nittis 23 - 70126 Bari

Figura 1.11: Una tabella non normalizzata

da, Civico, CAP e Località. Questa forma è importante perché agevola le ricerche dei valori in questo campo, permettendo di ricercare solo valori atomici. La Figura 1.12 ce ne mostra la versione 1NF.

tabella Soggetti
normalizzata
nella prima forma

Column Name	Data Type	Allow Nulls
IdSoggetto	int	☐
Cognome	varchar(50)	☑
Nome	varchar(50)	☑
CodiceFiscale	varchar(16)	☑
Toponimo	varchar(50)	☑
DenominazioneUrbanis...	varchar(150)	☑
Civico	varchar(50)	☑
CAP	varchar(5)	☑
Localita	varchar(50)	☑

Cognome	Nome	CodiceFiscale	Toponimo	Denominazione...	Civico	CAP
Vessia	Vito	VSSVTI74M30A6...	Via	Giuseppe De Nittis	23	70126

Figura 1.12: Tabella normalizzata in 1NF

1.5.3 Seconda forma normale

Una tabella di un database si dice in seconda forma normale (2NF) quando è in prima forma normale e tutti i campi non chiave sono in dipendenza funzionale con l'intera chiave primaria e non con una parte di essa. Ad esempio, introduciamo, in Figura 1.13, la tabella Vendite. La sua chiave primarie è costituita dalla coppia CodProdotto e Anno. Inoltre sono presenti i campi TotalePezziVenduti e NomeProdotto. La tabella è in prima forma normale. Il campo TotalePezziVenduti dipende dall'intera chiave primaria perché ha senso indica-

PuntoInformatico Libri

re questo dato di vendita solo se riferito ad un prodotto per un certo anno e quindi è in dipendeza funzionale con la chiave primaria. Il campo NomeProdotto, invece, dipende solo da una parte della chiave primaria e cioè dal campo CodProdotto e dunque è in dipendenza funzionale solo con una parte della PK.

Vendite

	Column Name	Data Type	Allow Nulls
🔑	CodProdotto	int	☐
🔑	Anno	int	☐
	NomeProdotto	varchar(250)	☑
	TotalePezziVenduti	int	☑
			☐

Figura 1.13: Tabella vendite non in 2NF

Per normalizzarla in 2NF dobbiamo creare una tabella esterna Prodotti, con chiave primaria CodProdotto, aggiungere ad essa il campo NomeProdotto, eliminare il campo omonimo dalla tabella Vendite e trasformare il campo CodProdotto di Vendita in una foreign key verso Prodotti. La Figura 1.14 chiarisce l'intera operazione.

Figura 1.14: Tabella vendite in 2NF

1.5.4 Terza forma normale

Una tabella di un database si dice in terza forma normale (3NF) quando è in seconda forma normale e tutti i campi non chiave non sono in dipendenza funzionale transitiva. Spieghiamo questo con-

cetto con un esempio. La Figura 1.15 introduce la tabella Ordini. Essa ha come chiave primaria il campo CodOrdine ed è costituita dai campi DataOrdine, Importo, CodCliente, RagSocialeCliente. La tabella è di sicuro in 1NF e in 2NF; inoltre, i campi DataOrdine e Importo sono corretti perché non contengono informazioni multiple (1NF) e sono entrambi in dipendenza funzionale con la PK (2NF). CodCliente e RagSocialeCliente rispettano anch'essi la 1NF e la 2NF ma hanno una strana relazione tra loro, perché mentre CodCliente dipende solo da NumOrdine, RagSocialeCliente dipende da questo e anche da CodCliente. Si crea, dunque una dipendenza funzionale transitiva:
NumOrdine --> CodCliente --> RagSocialeCliente
Quindi NumOrdine è in dipendenza funzionale transitiva con RagSocialeCliente.

Figura 1.15: Tabella ordini non in 3NF

Per normalizzarla in 3NF dobbiamo creare una tabella esterna Clienti, con chiave primaria CodCliente, aggiungere ad essa il campo RagSocialeCliente, eliminare il campo omonimo dalla tabella Ordini e trasformare il campo CodCliente di Ordini in una foreign key verso Clienti. La Figura 1.16 chiarisce l'intera operazione.

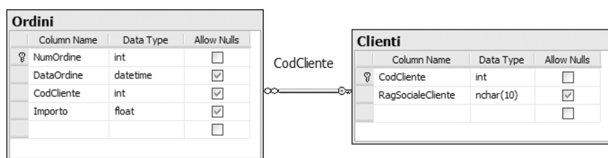


Figura 1.16: Tabella ordini non in 3NF

PuntoInformatico Libri

1.6 TIPI DI DATI

SQL Server gestisce tipi di dati diversi che possono essere classificati nel seguente modo:

- tipi di dati numerici;
- tipi di dati stringa;
- tipi di dati per data e ora;
- tipi di dati derivati;
- tipi di dati vari;
- tipi di dati definiti dall'utente.

1.6.1 Tipi di dati numerici

Tali valori, come da evidenza, vengono utilizzati per rappresentare i numeri. SQL Server definisce i seguenti tipi di dati numerici:

TIPO DI DATO	DESCRIZIONE
INT	Abbreviazione di INTEGER. Valori interi con segno memorizzabili in 4 byte (232)
SMALLINT	Valori interi con segno memorizzabili in 2 byte (2^{16})
TINYINT	Valori interi con segno memorizzabili in 1 byte (2^{8})
BIGINT	Valori interi con segno memorizzabili in 8 byte (2^{64})
DECIMAL(p,[s])	Valori a virgola fissa. L'argomento p (precision) specifica il numero totale di cifre comprese le cifre s (scala), dopo la virgola, partendo da destra.
NUMERIC(p,[s])	È un sinonimo di DECIMAL pertanto si comporta nello stesso modo
REAL	Valori in virgola mobile.
FLOAT(p,[s])	Valori in virgola mobile. La precisione del numero è definita da p.
MONEY	Usato per identificare i valori monetari. Corrisponde al DECIMAL a 8 byte.
SMALLMONEY	Corrisponde al tipo MONEY ma viene memorizzato in 4 byte

1.6.2 Tipi di dati stringa

Il tipo stringa è certamente il più pesante e oneroso da gestire da parte di SQL Server. Esistono, in particolare, tre tipologie di stringhe: le stringhe di caratteri, le stringhe binarie e le stringhe di bit.

TIPO DI DATO	DESCRIZIONE
CHAR([n])	Rappresenta un stringa di n caratteri. Il valore massimo è 8000. Se n viene omesso si interpreta come CHAR(1). È da rilevare che anche se il campo non viene riempito con una stringa della dimensione indicata da n, il resto del campo viene riempito con caratteri vuoti.
CHARACTER([n])	È sinonimo di CHAR.
VARCHAR([n])	Rappresenta una stringa di lunghezza variabile n, sempre di massimo 8000 caratteri. In questo caso la stringa occupa la dimensione reale e non quella massima nel database.
CHAR VARYING([n])	È sinonimo di VARCHAR.
CHARACTER VARYING([n])	È sinonimo di VARCHAR.
NCHAR([n])	Si comporta analogamente al tipo CHAR, ma i valori vengono rappresentati e salvati in alfabeto UNICODE anziché ASCII. Si tratta di un alfabeto molto più esteso che ha, però, lo svantaggio di richiedere 2 byte per rappresentare ciascun carattere, portando la dimensione massima a 4000 caratteri.
NVCHAR([n])	Si comporta come il tipo VARCHAR, ma usa la rappresentazione UNICODE.
TEXT([n])	Definisce una stringa a lunghezza fissa fino a 2 gigabyte. Ne viene sconsigliato l'uso nella versione 2005 dopo l'introduzione dei tipi VARCHAR(MAX).
NTEXT([n])	Valgono le stesse considerazioni fatte per TEXT, ma usando il formato UNICODE che quindi porta il numero più alto rappresentabile a 230--1. Ne viene sconsigliato l'uso nella versione 2005 dopo l'introduzione del tipo NVARCHAR(MAX).
VARCHAR(MAX)	È una novità di SQL Server 2005 e introduce un supporto esteso ai LOB (large objects), cioè ad unità di informazione di grandi dimensioni. L'opzione MAX consente di estendere grandemente la dimensione massima supportata dal tipo VARCHAR portandola fino a 230—1 byte. La gestione di questo campo è identica a quella dei campi VARCHAR tradizionali e quindi non impone l'uso del campo TEXT il cui uso, infatti, da questa versione viene deprecato.
NVARCHAR(MAX)	Valgono le stesse considerazioni fatte per il campo VARCHAR(MAX) con la differenza che le stringhe memorizzate sono in formato UNICODE. Questo influisce naturalmente sulla dimensione massima che, rispetto al tipo precedente, si dimezza.

PuntoInformatico | Libri

1.6.3 Tipi di dati data e ora

SQL Server supporta i formati data che vengono salvati nella loro rappresentazione numerica come numero di giorni passati dal 1 gennaio 1753 (DATETIME) o 1 gennaio 1990 (SMALLDATETIME). La data massima rappresentabile è il 31 dicembre 9999 con il DATETIME e un più timido 6 giugno 2079 con lo SMALLDATETIME. L'ora, invece, occupa un altro campo di 4 o 2 caratteri (rispettivamente per DATETIME e SMALLDATETIME) e rappresenta rispettivamente il numero di centesimi di secondo o di minuto passati dalla mezzanotte. L'impostazione predefinita di rappresentazione della data è nel formato "mm gg yyyy hh:mm AM" o "mm gg yyyy hh:mm PM" (Agosto 30 1974 10:45).

TIPO DI DATO	DESCRIZIONE
DATETIME	Specifica una data un'ora e viene rappresentato in un numero intero di 4 byte.
SMALDATETIME	Specifica una data un'ora e viene rappresentato in un numero intero di 2 byte.

1.6.4 Tipi di dati binari

È possibile conservare, all'interno di SQL Server, informazioni binarie pure senza interpretazione. Queste informazioni vengono memorizzate separatamente dagli altri tipi di valori gestibili dal database. Se in una tabella sono presenti più campi di tipo binario, il loro valore viene conservato in forma aggregata.

TIPO DI DATO	DESCRIZIONE
BINARY([n])	Permette di memorizzare un'informazione binaria di lunghezza fissa di esattamente n byte. Possono essere contenuti fino ad 8000 byte.
VARBINARY([n])	Conserva lo stesso tipo di informazione del tipo BINARY e della stessa massima dimensione, ma la sua dimensione può essere variabile e viene dunque occupato solo lo spazio realmente necessario all'informazione da memorizzare.

IMAGE([n])	Può memorizzare informazioni binarie di dimensione virtualmente illimitata (fino a 4 GB). La sua lunghezza è fissa ed è esprimibile in byte (parametro n). Tecnicamente è identico al campo TEXT, con la differenza che mentre quest'ultimo è in grado di conservare informazioni stampabile, IMAGE fa altrettanto per i dati binari puri e quindi per i caratteri anche non stampabili.
BIT	Conservare un singolo bit di informazione e quindi può essere usato per conservare dati boolean permettendo i valori true, false e null. Questo tipo di campo non è indicizzabile.
VARBINARY(MAX)	È uno dei nuovi campi LOB introdotti da SQL Server 2005. Analogamente a quanto avviene per il VARCHAR([n]) e il VARCHAR(MAX), il VARBINARY(MAX) consente di evitare l'uso del campo IMAGE([n]) consentendo, per qualsiasi esigenza, di optare sempre per i VARBINARY, semplificando così il modello di progettazione e di programmazione del database.

1.6.5 Tipi di dati speciali

Esistono alcuni tipi di campi speciali che non sono ascrivibili a categorie specifiche, pertanto vengono brevemente riassunti di seguito.

TIPO DI DATO	DESCRIZIONE
TIMESTAMP	Si tratta di un campo VARBINARY(8) o BINARY(8) a seconda se il campo accetta valori null oppure no. Il sistema mantiene un marcatore numerico incrementale per ciascun database. Ogni volta che viene effettuata una qualsiasi operazione di scrittura sul database, questo marcatore viene incrementato assegnando, così, all'operazione un progressivo numerico univoco. Per tale ragione il TIMESTAMP viene utilizzato per risalire in modo inequivocabile alla data e all'ora precisa di ciascuna operazione effettuata sul database.
SYSNAME	Si tratta di un campo NVARCHAR(128) che viene gestito in modo speciale perché contiene nomi di oggetti del database presenti nel catalogo di sistema.
CURSOR	Questo tipo di campo consente di dichiarare variabili che fanno riferimento a cursori. I cursori verranno discussi successivamente nel capitolo relativo al linguaggio T-SQL.

PuntoInformatico | Libri

TIPO DI DATO	DESCRIZIONE
UNIQUEIDENTIFIER	Si tratta di un numero identificativo assolutamente univoco, non solo nell'ambito della tabella o del database in cui viene creato e nemmeno soltanto a livello di motore SQL Server o di macchina, ma addirittura a livello assoluto. Esso, infatti, si basa sull'algoritmo Global Unique Identifier (Guid) di Windows. Questo tipo è piuttosto oneroso come richiesta, per essere semplicemente un contatore, infatti occupa ben 16 byte e va dunque usato solo se si necessità di identificativi assolutamente univoci a livello assoluto.
SQL_VARIANT	Questo tipo può contenere (temporaneamente) al suo interno qualsiasi tipo di valore. Logicamente ricorda il Variant di COM, tanto caro ai programmatori Visual Basic (non .NET). È un tipo lento ed esoso, nonostante sia molto comodo e pertanto va usato solo se strettamente necessario. Del resto introduce una forte de-tipizzazione nella struttura della tabella, rendendola così più difficile da comprendere.
TABLE	Il tipo TABLE è molto strutturato perché consente di contenere l'intero risultato di una query e quindi una matrice di righe e colonne restituite da un'interrogazione. Il suo uso verrà approfondito di seguito.
XML	SQL Server 2005 estende il supporto al formato XML già introdotto nella precedente versione. In seguito verrà trattato nello specifico questo aspetto. Invece, per quanto riguarda questo tipo di campo, si può brevemente dire che in SQL Server è possibile salvare isole di dati XML in campi del database.

1.6.6 Tipi di dati definiti dall'utente

In SQL Server 2005 è possibile creare tipi di dati personalizzati di tipo UDT (User Defined Data). Tale operazione è possibile grazie alla potente integrazione del .NET Framework all'interno della nuova versione del motore di accesso i dati.

1.6.7 Valori NULL

Un valore null è un valore speciale che può essere assegnato ad una co-

lonna. Di solito questo valore viene usato quando le informazioni della colonna mancano o non sono applicabili perché si tratta di un valore non previsto. Ad esempio, in una tabella di anagrafiche di soggetti, è facile che non si disponga del numero di telefono o dell'indirizzo email del soggetto; in tal caso nei campi relativi viene indicato il valore null. Null non rappresenta zero, una stringa di caratteri vuota o una stringa fatta di spazi vuoti o una stringa di lunghezza zero (alla C). Il Null è un valore completamente diverso ed esprima una condizione ben precisa e non ambigua. Infatti uno zero numerico può indicare sia l'assenza di un valore numerico nel campo e sia davvero un valore a zero e dunque come distinguere il primo caso dal secondo? Ci pensa il null. E così zero vorrà dire proprio zero e null vorrà dire che non si dispone della conoscenza di quel valore numerico. Per le stringhe il discorso è un po' più complicato perché, dal punto di vista umano, una stringa vuota o a lunghezza zero appaiano come l'equivalente di un valore che non c'è. Però come fare a distinguere, ancora una volta, il fatto che il la stringa sia davvero vuota dal fatto che semplicemente non se ne conosca il valore? Il null serve al secondo proposito. Il più grande inconveniente dei null è l'effetto negativo che hanno sulle operazioni matematiche. Tutte le operazioni che contengono un null hanno come risultato ancora un null. Il presupposto logico dietro questo comportamento, che a molti può risultare strano se non addirittura irritante, è che il null non rappresenza il valore vuoto (lo zero o la stringa vuota), ma l'indeterminazione del valore, pertanto qualsiasi operazioni matematica coinvolta dal null dà come risultato a sua volta una condizione di indeterminazione e quindi un null. Ecco alcuni esempi:

```
5 + (4 – 3) * 2 = 7
Null + (4 – 3) * 2 = Null
5 + (Null – 3) * 2 = Null
5 + (4 – Null) * 2 = Null
5 + (4 – 3) * Null = 7
```

Lo stesso effetto lo si ottiene all'interno delle query SQL, come vedre-

PuntoInformatico | Libri

mo nei capitoli successivi. Nemmeno le stringhe, come già accennato, sono estranee al problema: se si concatenano più stringhe ed una delle parti da concatenare è null, tutta stringa risultante sarà null. Dunque, bisognerà sempre scontrarsi con questo bizzarro valore e gestirlo in modo opportuno. Sempre.
È possibile impedire l'uso di null al momento della definizione delle tabelle, indicando espressamente l'opzione NOT NULL sulla colonna su cui impedirne l'uso. Se, nella definizione di un campo non contiene specificatamente l'autorizzazione o la negazione all'uso del null, SQL Server si comporta nel seguente modo:

- assegna null se l'opzione ANSI_NULL_DFLT_ON del comando SET è impostata su ON o se l'equivante stored procedure di sistema sp_dboption ha valore true (sp_dbption, 'ANSI_NULL_DFLT_ON', TRUE);
- assegna not null se l'opzione ANSI_NULL_DFLT_OFF del comando SET è impostata su ON o se l'equivante stored procedure di sistema sp_dboption ha valore true (sp_dbption, 'ANSI_NULL_DFLT_OFF', TRUE).

Esiste un'altra opzione che influenza le operazioni di concatenazione di stringhe contenenti parti null. Si tratta del comando SET CONCAT_NULL_YIELDS_NULL che, se impostato ad ON, forza a null il risultato della concatenazione contenente almeno un null, diversamente tratta il null come una stringa vuota e il risultato della concatenzione sarà dato dalla concatenazione di tutte le sottostringhe non null. Ad esempio:

```
SET CONCAT_NULL_YIELDS_NULL OFF
print 'Vito' + null + ' Vessia'
> Vito Vessia

SET CONCAT_NULL_YIELDS_NULL ON
print 'Vito' + null + ' Vessia'
> (NULL)
```

INSTALLAZIONE E AMMINISTRAZIONE

Dopo tanta teoria è finalmente giunto il momento di osservare il prodotto più da vicino. Innanzitutto è bene comprendere che esistono diverse versioni di SQL Server 2005. Tutte le versioni condividono lo stesso formato di memorizzazione fisica del database e si espongono all'esterno con lo stesso protocollo, garantendo così il massimo della scalabilità: infatti, nel momento in cui si ha l'esigenza di potenza o requisiti aggiuntivi rispetto alla versione del prodotto usata, è sufficiente acquistare ed installare la versione successiva per sfruttare le nuove e accresciute caratteristiche senza modificare nulla del codice di programma.

2.1 LE EDIZIONI DI SQL SERVER 2005

Proviamo ad elencare tutte le edizioni in ordine crescente di funzionalità:

- SQL Server 2005 Express Edition è la versione introduttiva e gratuita del prodotto e pertanto può essere scaricata, installata ed utilizzata senza alcun costo per lo sviluppatore e per l'utente finale dell'applicazione che sfrutta questa versione del database;
- SQL Server 2005 Workgroup Edition è una novità rispetto alla precedente versione 2000 ed è l'edizione entry level a pagamento che soddisfa la nicchia di mercato delle piccole aziende; in realtà questa è l'unica versione del prodotto, che pur recando il brand commerciale della versione 2005, è ancora basato sul motore della precedente versione SQL Server 2000 solo leggermente rivista;
- SQL Server 2005 Standard Edition è l'edizione di riferimento da sempre del prodotto perché è un'edizione completa a cui mancano solo le funzionalità Enterprise;

PuntoInformatico | Libri

- SQL Server 2005 Enterprise Edition è la versione top della gamma.

Esiste poi un'ulteriore versione, la Developer Edition, che ha le stesse funzionalità della versione Enterprise ma è utilizzabile per sviluppare le applicazioni e dunque non è utilizzabile in un contesto di produzione. Essa si installa anche su sistemi operativi desktop quali Windows XP Home Edition o superiori e Windows Vista Home Basic o superiori.
Osserviamo la seguente tabella riassuntiva dei requisti di ciascuna versione:

REQUISITI	Express	Workgroup	Standard ed Enterprise
Sistema operativo	Windows XP (tutte le versioni) Windows Server 2003 (tutte le versioni) Windows Vista Home Basic	Windows XP Professional Windows XP Media Center Windows XP Tablet Windows 2000 Professional	Windows Server 2003 Standard Edition Windows Server 2003 Enterprise Edition Windows Server 2003 Datacenter Edition Windows Small Business Server 2003 (tutte le versioni) Windows 2000 Server (tutte le versioni server)
Memoria minima	128 MB	512 MB	
Memoria consigliata	Da 512 MB in su	Da 1 GB in su	
Occupazioni su disco	350 MB per il programma e 390 MB per i database d'esempio		

Le varie versioni presentano evidentemente anche differenza di funzionalità e di limiti da un'edizione all'altra. La tabella successiva ne elenca le principali differenze:

REQUISITI	Express	Workgroup	Standard	Enterprise
Numero di CPU massimo	1	2	4	Nessun limite
Memoria centrale massima	1 GB	3 GB	Nessun limite	Nessun limite
Supporto ai 64 bit	Windows on Windows	Windows on Windows	SI	SI
Dimensione del database	4 GB	Nessun limite	Nessun limite	Nessun limite
Partizionamento	NO	NO	NO	SI
Indicizzazione parallele	NO	NO	NO	SI
Viste indicizzate	NO	NO	NO	SI
Mirroring	NO	NO	SI	SI
Clustering di failover	NO	NO	SI	SI
Modifiche del sistema in linea	SI	SI	SI	SI
Ripristino in linea	NO	NO	NO	SI
Management Studio Express	SI	SI	SI	SI
Management Studio	NO	SI	SI	SI
Ottimizzazione guidata database	NO	NO	SI	SI
Miglioramenti della facilità di manutenzione	SI	SI	SI	SI
Ricerca full-text	NO	SI	SI	SI
SQL Agent	NO	SI	SI	SI
Stored procedure, trigger e viste	SI	SI	SI	SI
Miglioramenti di T-SQL	SI	SI	SI	SI
Integrazione .NET	SI	SI	SI	SI
Tipi definiti dall'utente	SI	SI	SI	SI
XML nativo	SI	SI	SI	SI
XQuery	SI	SI	SI	SI
Notification Services	SI	SI	SI	SI
Importazione/esportazione	SI	SI	SI	SI
Integration Services con trasformazioni di base	NO	NO	SI	SI
Integration Services con trasformazioni avanzate	NO	NO	NO	SI
Replica	SI	SI	SI	SI
Servizi Web (endpoint HTTP)	NO	NO	SI	SI
Server di report	SI	SI	SI	SI
Generatore di report	NO	SI	SI	SI
Funzioni di analisi SQL	SI	SI	SI	SI
Data warehousing	NO	NO	SI	SI
Analysis Services	NO	NO	SI	SI
Data mining	NO	NO	SI	SI

PuntoInformatico Libri

2.2 INSTALLAZIONE DEL PRODOTTO

Il processo di installazione di SQL Server è un'attività che viene sempre più semplificata con l'evolvere delle versioni. La procedura di installazione di SQL Server 2005 introduce uno strumento più efficace di verifica dei requisiti rispetto alle precedenti versioni. Di seguito verrano descritte, passo per passo, le fasi di installazione di SQL Server 2005 Developer Edition in lingua inglese che però non è troppo dissimile dall'installazione delle altre versioni, compresa la Express. In Figura 2.1 è possibile osservare la dialog che si presenta all'inserimento del DVD di installazione del prodotto.

Figura 2.1: Si parte

Abbiamo già precisato che l'edizione Developer corrisponde a livello di funzionalità alla Enterprise, dunque è completa. Inoltre dal DVD è possibile installare la versione X86 per processori e sistemi operativi a 32 bit, la versione X64 (per processori che supportano i set estesi di istruzioni AMD64 o Intel EMT-64 e per la corrispondente versione di Windows) e Itanium. Una volta effettuata la scelta, si presenta una seconda dialog del wizard di installazione che offre

una serie di opzioni, come mostrato in Figura 2.2.

Figura 2.2: Istallazione dei prerequisiti

Se sulla macchina non è stata ancora installata nessuna versione di SQL Server 2005, nemmeno l'edizione Express in automatico da qualche altro prodotto, allora si proceda prima ad installare la voce "Run the SQL Native Client Installation Wizard" che installare le librerie client per la corretta esecuzione di tutte le parti client e degli strumenti di amministrazione di SQL Server. Questa operazione sarebbe in realtà non necessaria installando direttamente la voce "Server components, tools, Books Online, and samples", che si occupa di installare i componenti server, però si sono riscontrati a volte alcuni problemi di concorrenza di installazione delle SQL Native Client insieme ad altri componenti e quindi è opportuno procedere in modo separato. Successivamente, al termine dell'installazione, riprendere il setup e selezionare la voce "Run the SQL Native Client Installation Wizard". A questo punto parte il wizard di setup delle parti server. Dopo un paio di schermate di riepilogo, viene eseguito il System

PuntoInformatico | Libri

Confuration Check, come mostrato in Figura 2.3.

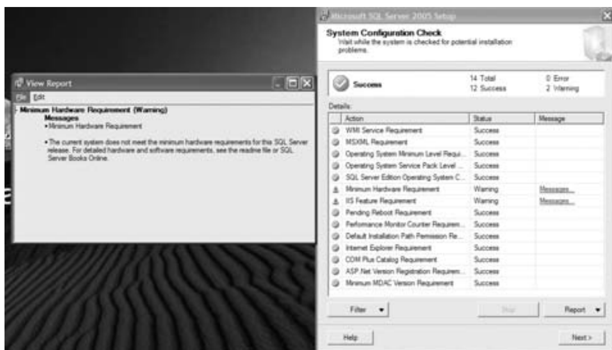


Figura 2.3: Il sytem Confuration Check

Esso effettua una scansione delle caratteristiche hardware e software del sistema ospite e ne verifica il soddisfacimento dei requisiti. L'elenco completo delle voci esaminate è osservabile in figura e va dalla versione del sistema operativo installato, alla memoria centrale del sistema ad altri requisiti software molto specifici (ad esempio la versione di XML e di MDAC installate). Tipicamente una macchina con Windows XP Service Pack 2 e Windows Vista (solo per le versioni di prodotto che supportano questo sistema operativo desktop) o Windows Server 2003 Service Pack 1, magari con Internet Explorer 7 già installati, dovrebbero disporre di tutti i requisiti sufficienti per l'installazione di SQL Server 2005. Ciascun voce di verifica può avere come esito Success (con un rassicurante bollino verde), Warning (con un'icona gialla) che indica una non perfetta aderenza ai requisiti che però non pregiudica il proseguo dell'installazione e Error che invece è un problema bloccante per l'installazione. In quest'ultimo caso l'installazione va interrotta e va ripresa solo dopo aver soddisfatto il requisito. Per ciascuna voce è possibile ottenere una descrizione precisa e più estesa per una maggiore comprensione.

Se tutti i requisiti sono soddisfatti o quanto meno sono solo presenti al più Warning, si può proseguire con l'installazione premendo Next. A questo punto, a seconda delle versioni e del tipo di supporto di installazione che si ha disposizione, può essere richiesto l'inserimento del CD Key del prodotto oppure questo può essere implicitamente presente nel setup. Successivamente si giunge alla sezione "Components to Install" del wizard, come mostrato in Figura 2.4. Da esso è possibile selezionare i servizi e i componenti client e server da installare, ciascuno inserito in una checkbox di selezione. Troviamo la voce "SQL Server Database Services" per l'installazione del motore RDBMS vero e proprio, "Analysis Services" per i servizi OLAP e di data wharehousing, "Reporting Services" che è il nuovo servizio di reportistica web integrato in SQL Server, "Notification Services" per la gestione degli eventi di database, "Integration Services", il nuovo sistema di importazione e migrazione dei dati che sostituisce il vecchio DTS (Data Transformation Services di SQL Server 7 e 2000) e le "Workstation components, books online and development tools". È chiaramente possibile installare più prodotti contemporaneamente o anche tutti insieme. Nella stessa dialog è presente il tasto "Advanced" che permette di impostare a basso livello le successive opzioni di installazione di SQL Server. In realtà, le opzio-

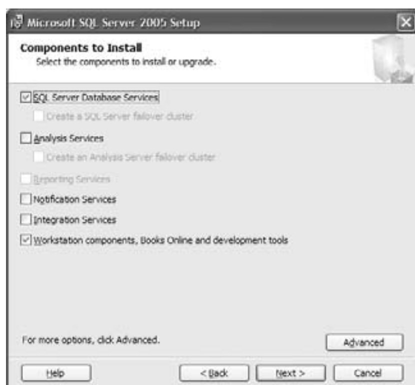


Figura 2.4: Scelta dei componenti da installare

ni di default sono generalmente valida in quasi tutti i casi, ma se si intende effettuare un più preciso tuning dell'applicazione è necessario optare per la soluzione avanzata.
A questo punto si viene condotti alla tipica dialog dei setup basate su Windows Installer, da cui è possibile selezionare o deselezionare parti da installare e anche indicare se queste dovranno essere da subito copiato nel sistema dal DVD, se dovranno essere assenti o se dovranno essere copiate alla prima richiesta. Per un migliore fruizione degli esempi presenti in questo volume è consigliabile installare i database di esempio.

2.2.1 Le istanze di SQL Server

A questo punto viene richiesto il nome dell'istanza di SQL Server da installare. È infatti possibile installare più server di database SQL Server nella stessa macchina. Ciascun motore, detto istanza, avrà un nome separato e persino una diversa directory di installazione e sarà dunque completamente indipendente da tutti gli altri, esattamente come se fosse installato su una macchina diversa. Il nome del motore di database SQL Server si identifica con il nome Windows della macchina su cui è installato. Perciò, il motore SQL Server di un server chiamato \\MioServer sarà proprio MioServer. Tuttavia, in presenza di più istanze di SQL Server sulla stessa macchina, ciascuna di essa avrà un nome differente (una named instance). Eccone la sintassi:

```
NomeHost/NomeIstanza
```

Ad esemio MioServer/Istanza1 oppure ServerDati/Contabilità, ecc.. L'istanza detta di default della macchina può omettere la parte NomeIstanza e chiamarsi direttamente come l'host. Il setup, dunque, richiede se installare o un'istanza di default (sempre che non ve ne siano altre già presenti) o una named, in tal caso ne viene richiesto il nome, come mostrato in Figura 2.5.

Figura 2.5: Scelta d'ellistanza da installare

Tramite il tasto Installed instances è possibile conoscere l'elenco delle istanze di SQL Server già installate nel sistema. Si consideri che sulla stessa macchina possono convivere istanze diverse e contemporanee di SQL Server 2000 e SQL Server 2005.

2.2.2 Servizi e modalità di autenticazione

Premendo Next si giunge alla dialog "Service Account". Ciascuno dei server di SQL Server (RDBMS, Analysis, Agent, ecc...) viene installato nel sistema come Servizio Windows, che è una particolare modalità di installazione ed esecuzione non interattiva di Windows che consente di fare girare le applicazioni anche se nessun utente si è loggato a Windows e, più in generale, con un livello di autorizzazione solitamente più elevato rispetto a quello consentito agli utenti. Esiste un utente di sistema di default per l'esecuzione dei servizi: si tratta di Local System Account, tuttavia è possibile far girare i servizi con un utente interattivo. La dialog corrente ci consente proprio di impostare queste opzioni per ciascuno dei servizi che SQL Server andrà ad installare. Si consiglia di lasciare l'utente Local System. Da

PuntoInformatico Libri

qui è inoltre possibile impostare l'avvio automatico di alcuni servizi al termine del setup.

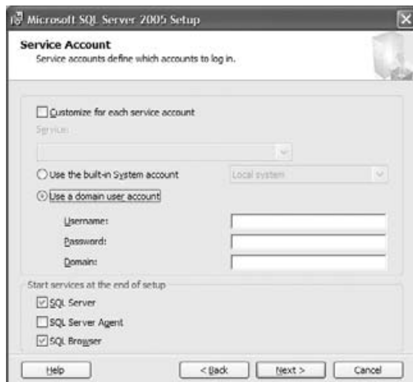


Figura 2.6: Opzione di esecuzione dei servizi

Successivamente, dopo il Next, si accede alla pagina "Authentication Mode" del wizard di installazione. SQL Server consente l'autenticazione degli utenti in due modalità:

- Windows Authentication (è la modalità preferite a consigliata da Microsoft perché direttamente integrata con Windows: gli utenti di SQL Server sono esattamente gli utenti di Windows per cui non vi è una fase esplicita di login per accedere a SQL Server, ma questi prende l'utente e le grant in maniera automatica ed integrata direttamente dall'utente di Windows che sta tentanto l'accesso);
- SQL Server Authentication (rappresenta la modalità nativa e più antica di SQL Server per gestire gli utenti: essi sono direttamente definibili all'interno di SQL Server in tabella di sistema e pertanto la fase di autenticazione al motore richieste esplicitamente l'indicazione di username e password di utenti nativi di SQL Server; in questa modalità è inoltre necessaria la presenza di un utente am-

ministratore e quindi col grado più alto di accesso al sistema, tale utente si chiama obbligatoriamente "SA", che è l'acronimo di System Account).

Nella dialog corrente è possibile scegliere se installare SQL Server esclusivamente in modalità Windows Authentication o in modalità mista (Mixed Mode): in questo secondo caso sarò possibile accedere al server di database in entrambe le modalità. Inoltre, in mixed mode sarà necessario indicare la password dell'utente SA. In Figura 2.7 è possibile osservare la dialog.

Figura 2.7: Scelta della modalità di autenticazione

2.2.3 Opzioni di collation

La collation è un'importante impostazione di SQL Server e in generale dei database perché determina non solo il set di caratteri in uso, che dipende dalla lingua, ma anche il comportamento nella ricerca e negli ordinamenti dei caratteri maiuscoli e minuscoli (se "A" deve essere uguale ad "a") ed alle lettere accentate (se "è" deve essere uguale ad "é" e a "e") nei campi CHAR e VARCHAR. Queste proprietà determinano, dunque, se il set di caratteri è case sensitive e/o

accent sensitive.
Per impostazione predefinita, il campo eredita le impostazioni a livello di database definite in fase di creazione dello stesso, o modificato successivamente con il comando ALTER DATABASE. Eccone un esempio:

```
USE Tempdb
GO
ALTER DATABASE Pubs COLLATE  COLLATE LATIN1_GENERAL_CS_AI
GO
```

Innanzitutto bisogna precisare che non è possibile cambiare la collation di un database dall'interno del database stesso ma bisogna spostarsi su un altro per eseguire la variazione. Nell'esempio si usata TempDb, il database temporaneo di SQL Server, per poi eseguire l'alterazione. Se in fase di creazione del database non viene fatta la scelta della collation, verrà ereditata quella valida per l'istanza che è stata definita in fase di setup.
Ciò premesso è possibile eseguire ordinamenti/ricerche case sensitive o case insensitive, accent sensitive o accent insensitive a prescindere da quale sia la collation del campo su cui si esegue l'operazione.
A questo scopo si utilizza la clausola COLLATE e, qualunque sia l'impostazione della colonna, l'istruzione che segue

```
SELECT *
FROM Orders
WHERE CustomerId COLLATE LATIN1_GENERAL_CI_AI = 'FOLIG'
```

eseguirà un cast del campo Nome restituendo non solo i valori "FOLIG" ma anche le varianti "Fòlig", "Folìg", ecc.

I suffissi "CS" e "AI" che indicano rispettivamente una collation Case Sensitive e Accent Insensitive.

In questa fase del setup viene proprio definito questo aspetto, come si può osservare in Figura 2.8.

Figura 2.8: Scelta della collation

Fino a SQL Server 2000 era definito un elenco di Collation standard, ciascuna delle quali era identificata da un nome e portava con se sia la lingua che l'opzione di case sensitive e accent sensitive. Dunque bastava impostare una di questi nomi per definire tutte le proprietà. La più gettonata, non che quella proposta di default al momento dell'installazione, è la LATIN1_GENERAL_CI_AI: essa definisce l'alfabeto latino e quindi è ottimale nei paesi occidentali, Italia compresa, è case insensitive e accent insensitive. Quando si installa SQL Server 2005, come si può vedere dall'immagine, sarà ancora possibile esprimere queste opzioni attraverso il nome di collation, per compatibilità con eventuali database SQL Server 2000 preesistenti, oppure usare il nuovo “Collation designator and sort order” per specificare ogni aspetto della collation e del sort order.

2.2.4 Completamento dell'installazione

A questo punto, dopo il Next, apparirà una pagina di riepilogo delle op-

PuntoInformatico Libri

zioni e dei componenti da installare, la sezione “Ready to Install”. Premendo ancora Next, si arriva alla pagina del wizard “Setup Progress”, come mostrato in Figura 2.9 che eseguirà l'installazione vera e propria.

Figura 2.9: Pronti per l'installazione

La pagina mostra l'elenco dei componenti in corso di installazione e la relativa percentuale di completamento. È interessante notare che l'installazione di alcuni componenti procede in parallelo e così via fino a quando tutti i componenti non saranno stati installati correttamente. Ancora una volta, eventuali errori durante l'installazione verranno segnalati in modo puntuale. Il processo di installazione, dipendentemente dalla quantità di componenti da installare e dalle caratteristiche della macchina, può durare anche più di mezz'ora, tempo durante il quale non si potrà che attendere religiosamente l'esito. Se tutto sarà andato per il meglio, finalmente un'ultima dialog di segnalerà l'avvenuta corretta installazione e saremo pronti ad usare SQL Server 2005.

2.3 SQL SERVER MANAGEMENT STUDIO

SQL Server Management Studio (per brevità lo chiameremo Management Studio) è lo strumento principale di amministrazione, controllo, pro-

grammazione ed uso di SQL Server 2005 ed è pertanto indicato per tutte le tipologie di utenti: amministratori di database, progettisti, sviluppatori e utenti finali. Esso, infatti, a differenza del passato è un unico strumento integrato e sostituisce tutti i precedenti strumenti forniti con SQL Server 2000 che rendevano la gestione più caotica e frammentaria. È richiamabile dal menù Programmi di Windows alla voce Microsoft SQL Server 2005 --> SQL Server Management Studio. La Figura 2.10 mostra la dialog di login che appare all'avvio del programma. Esso consente l'autenticazione a SQL Server, in modalità SQL Server o integrata, perché ogni operazione che si effettuerà su SQL Server attraverso Management Studio, viene comunque effettuata sempre attraverso comandi T-SQL eseguiti dall'utente con il quale ci si è autenticati a SQL Server attraverso Management Studio. È interessante ai fini didattici, infatti, attivare SQL Server Profiler, lo strumento che permette di tracciare ogni comando SQL lanciato su SQL Server, e osservare quali comandi T-SQL esegue Management Studio in corrispondenza di operazioni che l'utente effettua dallo strumento grafico. Pertanto, ogni operazione effettuabile a Management Studio è anche effettuale direttamente con comandi T-SQL. A differenza di quanto avveniva con MSDE, l'equivalente per SQL Server 2000 di SQL Server 2005 Express, quest'ultima fornisce, anche se come download separato, una versione leggera ma ricca di

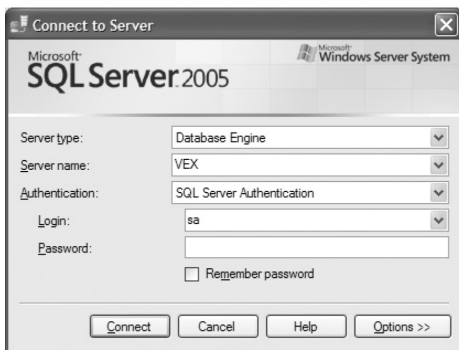


Figura 2.10: Login di accesso a SQL Server Management Studio

PuntoInformatico Libri

quasi tutte le funzionalità importati di SQL Server Management Studio, che si chiama SQL Server Management Studio Express. Quasi tutte le funzionalità descritte in questo paragrafo sono riferibili anche alla versione Express dell'ambiente di amministrazione.
La Figura 2.11 mostra come si presenta Management Studio all'avvio dopo il login. In basso a sinistra possiamo individuare l'area dei Registered Server, cioè dei server a cui ci si è registrati e che quindi è possibile gestire da Management Studio. È infatti possibile amministrare più istanze di SQL Server, presenti sulla macchina locale o su macchine remote. Nella parte alta dello schermo, sulla sinistra, troviamo l'Object Explorer che contiene l'albero di tutti gli oggetti presenti su un server, dal Database Engine al motore di reportistica ad Analysis Server ed ogni altro elemento del database. Sulla destra troviamo la finestra Properties che è contestualizzata, cioè il suo contenuto varia in base all'oggetto selezionato (ad esempio un server, una tabella, un campo o ogni altro oggetto di SQL Server). Fondamentalmente è un editor di proprietà. L'elemento centrale dell'applicazione è la finestra Document che può contenere l'editor delle query o le finestre di navigazione e gestione degli elementi di SQL Server (ad esempio un server, una tabella, un campo, ecc…). Da qui si effettua ogni modifica o gestione degli oggetti di SQL Server.

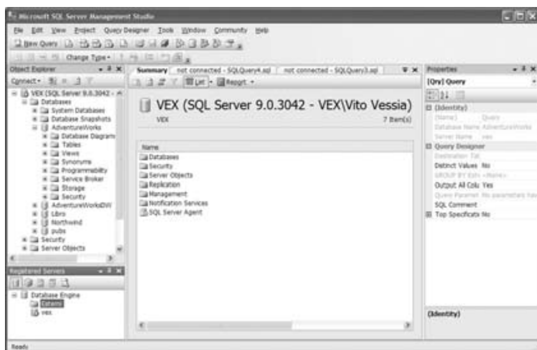


Figura 2.11: SQL Server Management Studio

È importante osservare che la posizione di ogni elemento non è fissa, ma ogni parte è flottante (dockable) e quindi si può spostare liberamente ad una parte all'altra della finestra principale, in modo personalizzato.

2.3.1 Registrazione dei server

La registrazione di un nuovo server è un'attività che permette di agganciare, autenticarsi e gestire un motore di database locale o remoto. La registrazione va effettuata una sola volta, dopo di che sarà sufficiente solo autenticarsi al server agli accessi successivi, ma Management Studio conserverà i riferimenti (indirizzo di rete o nome dell'host, nome dell'istanza, tipo di autenticazione, ecc...) di tutti i server registrati. La registrazione si effettua selezionando il nodo principale Engine nel pannello Registered Servers, poi invocandone il menù contestuale con il tasto destro del mouse ed infine scegliendo la voce New --> Server Registration. A questo punto apparirà una dialog come mostrato in Figura 2.12. Si può notare come sia possibile associare al server registrato un nome diverso da quello reale.

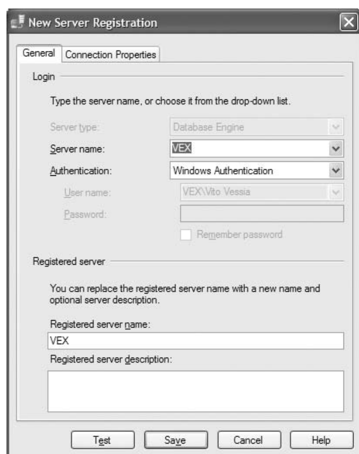


Figura 2.12: Registrazione di un nuovo server

In presenza di molti server registrati, sarà possibile ordinare questi server in modo gerarchico in gruppi logici gestibili in modo personalizzato. La creazione di un nuovo gruppo si effettua dal pannello Registered Servers (New --> Server Group). Da quel momento sarà possibile registrare nel nuovo gruppo altri server o creare dei sottogruppi.

2.3.2 Avvio e arresto di SQL Server

Da Management Studio è possibile arrestare o avviare un'istanza di SQL Server. È sufficiente selezionare da Registered Servers l'istanza o il nodo principale dell'istanza da Object Explorer, invocarne il menù contestuale e selezionare le autoesplicative voci Start, Stop, Pause, Resume o Restart.

2.4 QUERY EDITOR

Una delle principali ragioni per cui accedere a Management Studio sarà il suo Query Editor interno, cioè lo strumento per scrivere ed inviare query e comandi T-SQL di ogni tipo su ogni database registrato in Management Studio. Esso sostituisce il buon vecchio Query Analyzer, presente in SQL Server 7 e 2000, e avvicina la scrittura delle query molto di più allo sviluppo di codice in Visual Studio o comunque certamente di più di quanto non facesse il suo predecessore. Inoltre ha una fantastica funzionalità alla Office: il salvataggio automatico temporizzato sia di query già salvate dall'utente che di query nuove non ancora salvate. Alzi la mano chi non ha mai perso una query su cui ha lavorato per una buona mezz'ora, a causa di un crash e di un blackout. Un po' come accadeva per Office, tanti anni fa, prima che introducessero il salvataggio automatico...
Per lanciare una nuova query è sufficiente selezionare il tasto New Query. Se si è già connessi ad uno dei server registrati e questo è selezionato nell'Object Explorer, nella finestra Document apparirà nella parte alta un'area in cui digitare la query o il comando T-SQL e subito sotto l'area dei risultati in forma di griglia che mostra il resultset o come area di output dei messaggi di SQL Server come rispo-

sta al comando. Per eseguire il comando è sufficiente premere il tasto Execute della toolbar o premere F5. In Figura 2.13 è possibile osservare un esempio di query e la griglia del resultset. Il comando di esecuzione non esegue necessariamente l'intera query, se si seleziona solo una parte del codice SQL nel Query Editor, la successiva Execute eseguirà solo il codice selezionato.
Se il comando è costituito da più comandi, è possibile inserire tra ciascuno di essi il comando T-SQL GO in modo che SQL Server restituisca i risultati parziali, altrimenti restituirà i risultati tutti in una volta. Si osservi l'esempio di seguito riportato:

```
SELECT DISTINCT City, Country
FROM Suppliers
GO
SELECT DISTINCT City, Country
FROM Customers
GO
```

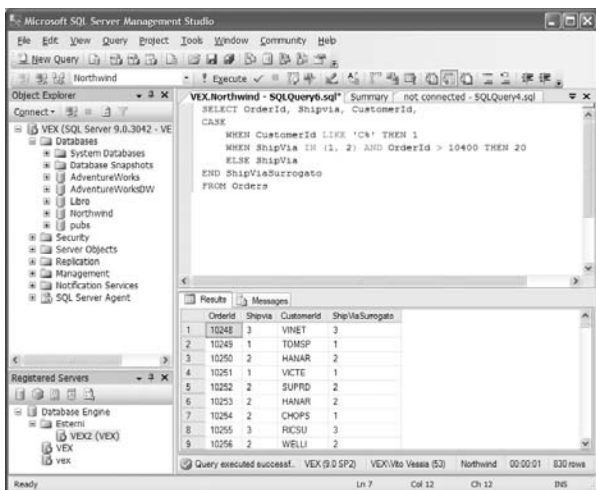


Figura 2.13: Il Query Editor di SQL Management Studio in azione

PuntoInformatico Libri

Il Query Editor offre una comoda funzionalità di syntax coloring, cioè viene riconosciuta la sintassi del comando T-SQL che viene editato e ciascuna parola viene colorata in modo opportuno, consentendo di distinguere le parole chiave del linguaggio, dai nomi degli oggetti del database, dai commenti, dalle stringhe e dalle funzioni.
La toolbar del Query Editor offre una miniera di interessanti funzionalità. La Figura 2.14 riporta la toolbar con i var pulsanti identicati da un numero. Osserviamone i più interessante facendo riferimento al numero riportato in figura:

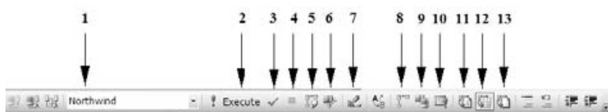


Figura 2.14: La toolbar del Query Editor

1 Available Databases (il campo indica il nome del database su cui verrà eseguito il comando T-SQL la combo riporta l'elenco dei database presenti nel server corrente);
2 Execute (esegue il comando correntemente presente ne Query Editor o la porzione di comando correntemente selezionata);
3 Parse (effettua una verifica sintattica del codice SQL del comando senza eseguire realmente il codice; utile ad esempio per testare la bontà sintattica i comandi di scrittura prima che vengano eseguiti);
4 Cancel Executing Query (questo tasto si attiva solo durante l'esecuzione di una query e, se premuto, invia una richiesta di arresta della query in corso);
5 Display Estimated Execution Plan (invoca l'Estimated Execution Plan, si legga la nota relativa all'Actual Query Plan per saperne di più);
6 Analyze Query in Database Engine Tuning Advisor (invoca lo stru-

mento di tuning del database che segnala eventuali ottimizzazioni da effettuare sul database per rendere più rapida la query eseguita, tipicamente proponendo la creazioni di indici specifici; la differenza tra l'Estimated e l'Actual è che mentre il primo viene definito senza realmente eseguire le query e quindi solo sulla base di un'analisi sintattica del comando e degli oggetti coinvolti, il secondo è decisamente più attendibile perché analizza il reale comportamento di SQL Server durante l'esecuzione del comando, infatti ne richiede necessariamente l'esecuzione per essere prodotto);

7 Design Query in Editor (consente di disegnare la query graficamente, come mostrato in Figura 2.15, fornendo la possibilità di sviluppare query anche molto complesse in un ambiente completamente grafico, senza essere esperti del linguaggio SQL);

8

Figura 2.15: L'editor visuale di Query

9 Include Actual Execution Plan (esegue il vero Execution Plan, cioè lo strumento di analisi e monitoraggio delle prestazioni delle singole query eseguite; analizza e scompone la query nei singoli passaggi necessari alla sua esecuzione da parte del Database Engine, dando a ciascun passaggio un peso in percentuale di

PuntoInformatico Libri

utilizzo di CPU e di accesso al disco, in modo che se ne possano comprendere e risolvere gli eventuali colli di bottiglia per migliorarne le prestazioni, ad esempio adottando indici);

10 Include Client Statistics (include, dopo l'esecuzione della query, una statistica completa dei tempi, dei pacchetti inviati in rete ed ogni altra risorse usata dal comando sul client durante l'esecuzione; questo strumento p utile per individuare eventuali colli di bottiglia sul client; la Figura 2.16 riporta un esempio);

11

Results | Execution plan | Client Statistics

	Trial 3	Trial 2	Trial 1	Average
Client Execution Time	08:02:55	08:02:36	08:02:20	
Query Profile Statistics				
Number of INSERT, DELETE and UPDATE statements	0 →	0 →	0 →	0.0000
Rows affected by INSERT, DELETE, or UPDATE statem...	0 →	0 →	0 →	0.0000
Number of SELECT statements	2 →	2 ↑	1 →	1.6667
Rows returned by SELECT statements	831 →	831 ↑	830 →	830.6667
Number of transactions	0 →	0 →	0 →	0.0000
Network Statistics				
Number of server roundtrips	3 →	3 ↑	1 →	2.3333
TDS packets sent from client	3 →	3 ↑	1 →	2.3333
TDS packets received from server	10 →	10 ↑	6 →	8.6667
Bytes sent from client	538 →	538 ↑	392 →	489.3333
Bytes received from server	32587 →	32587 ↑	22595 →	29256.3300
Time Statistics				
Client processing time	15 ↓	125 ↓	156 →	98.6667
Total execution time	15 ↓	156 →	156 →	109.0000
Wait time on server replies	0 ↓	31 ↑	0 →	10.3333

Figura 2.17: Le Client Statistics

12 SQLCMD Mode (esegue il comando usando l'utilità a riga di comando SQLCMD anziché direttamente; SQLCMD è il nuovo strumento di SQL Server 2005 che consente l'esecuzione di query e di comandi T-SQL direttamente da chiamate a linea di comando passando il comando T-SQL come parametro, è utile per l'esecuzione di operazioni batch schedulate o per avere accesso a SQL Server senza scrivere un programma tradizionale che apre la connessione ed esegua il comando);

13 Results to Text (se attivo, consente di visualizzare il risultato della query eseguita o del comando eseguito come testo non for-

mattato nell'area dei risultati, come osservabile in Figura 2.17);

14

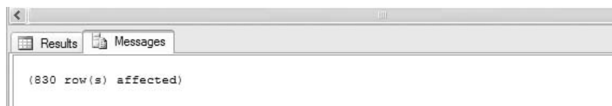


Figura 2.17: Il resultset mostrato in formato testuale

15 Results to Grid (se attivo, consente di visualizzare il risultato della query eseguita, sempre che questo restituisca un resultset, all'interno di una comoda griglia; è da rilevare che anche se è attiva questa modalità, ma il comando eseguito non produce resultset ma solo messaggi di SQL Server, tali messaggi vengono mostrati in modalità Result to Text, come mostrato in Figura 2.18);

16 Results to File (se attivo, consente di salvare il risultato della query in un file di testo esterno il cui path è impostabile dall'utente);

17

Results | Execution plan | Client Statistics

OrderId	Shipvia	CustomerId	ShipViaSurrogato
10248	3	VINET	3
10249	1	TOMSP	1
10250	2	HANAR	2
10251	1	VICTE	1
10252	2	SUPRD	2
10253	2	HANAR	2
10254	2	CHOPS	2
10255	3	RICSU	3
10256	2	WELLI	2
10257	3	HILAA	3
10258	1	ERNSH	1
10259	3	CENTC	3
10260	1	OTTIK	1

Figura 2.18: L'output di SQL server in risposta ad un comando

PuntoInformatico Libri

2.5 AMMINISTRAZIONE DI SQL SERVER

Attraverso SQL Server Management Studio è possibile gestire ogni aspetto di SQL Server e di agganciare i motori in versioni 2005, 2000 e 7. Una volta registrato un server, nell'Object Explorer si popolerà un nodo corrispondente al server registrato. In Figura 2.19 possiamo osservare una parziale espansione dei nodi di un server. Possiamo osservare un primo nodo Databases sotto il quale troviamo i database installati nel server oltre che i database di sistema che vedremo nel proseguo. Il nodo Security consente di gestire tutta la sicurezza del database: utenti, ruoli e accessi. I Server Objects e Replications riguardano degli aspetti avanzati di SQL Server che non tratteremo in questo volume. Management è la sezione più strettamente di profilazione e di controllo di SQL Server perché consente la consultazione dei log storicizzati e delle attività in esecuzione in tempo reale. SQL Server Agent è, infine, il potente schedulatore integrato di SQL Server che consente di pianificare e di eseguire operazioni programmate sul database e non solo, con un livello di controllo e di servizi offerti ai singoli agenti o ai singoli task da realizzare molto superiore, ad esempio, allo schedulatore integrato di Windows. Per

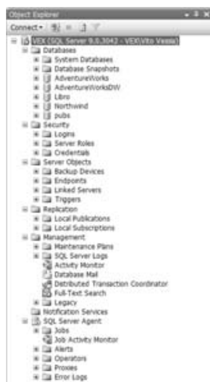


Figura 2.19: L'albero di gestione del server in Object Explorer

accedere, visualizzare e modificare le proprietà degli oggetti del database o del motore, che graficamente vengono rappresentati come nodi dell'albero dell'Object Explorer, è sufficiente selezionare il nodo interessato, richiamare il menù popup contestuale con il tasto destro del mouse e scegliere la voce Properties.

2.5.1 Configurazione e tuning del Database Engine

Al momento dell'installazione il setup esamina il sistema ed effettua un primo tuning, cioè una prima analisi del sistema ospite per impostare le corrette opzioni di esecuzione e di sfruttamento delle risorse da parte del Database Engine. Tuttavia è sempre possibile intervenire manualmente per verificare o modificare tali impostazioni. Questa operazione è possibile dall'Object Explorer selezionando il nodo relativo al server da configurare, richiamando il menù contestuale col tasto destro del mouse e scegliendo la voce Properties. Questa operazione fa apparire la dialog Server Properties organizzata per pagine. La prima, General (Figura 2.20), riassume le caratteristiche principali del motore: nome dell'istanza (Name), versione (SQL Server 2005 corrisponde alla versione 9.0.x, SQL Server 2000 corri-

PuntoInformatico Libri

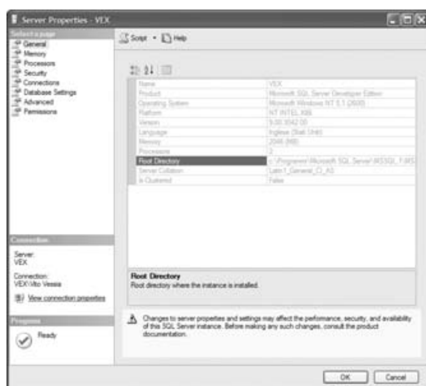


Figura 2.20: La scheda del server

sponde alla versione 8.0.x e SQL Server 7 corrisponde alla 7.0.x), edizione del prodotto (Product), sistema operativo host, piattaforma (X86, X64 o Itanium), lingua, numero di processori presenti nel sistema (per processore è da intendere il singolo core per processori multicore come AMD Athlon X2, Intel Core Duo e Intel Pentium D), collation di sistema e directory di installazione del motore.
Nella successiva pagina Memory, osservabile in Figura 2.21, è possibile effettuare il tuning di utilizzo della memoria: dalla minima e massima utilizzabile da SQL Server (Minimum e Maximum Server Memory), alla memoria utilizzabile nella creazione di indici e nell'esecuzione di query.

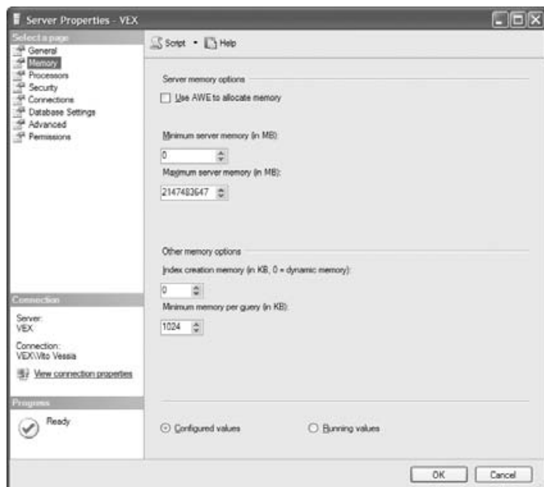


Figura 2.21: Gestione della memoria

Analogamente avviene per la pagina Processors (Figura 2.22). Per una migliore comprensione degli aspetti più avanzati legati alla gestione delle risorse del sistema ospite, si rimanda alla documentazione in linea (Books Online).

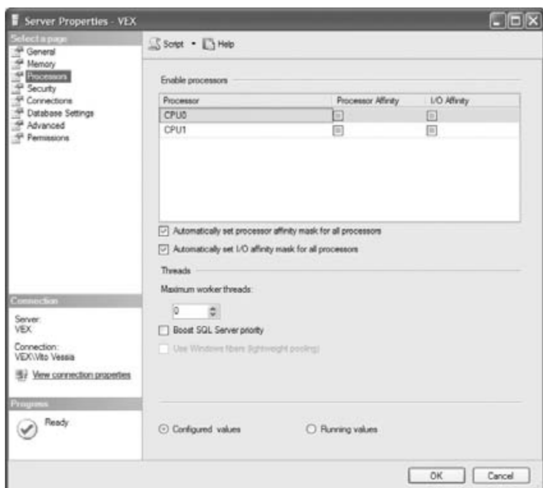


Figura 2.22: Gestione dei processori e dei core

2.5.2 Creazione di un database

La creazione di un nuovo database si effettua selezionando il nodo Databases del server in Object Explorer, richiamando il menù contestuale col tasto destro del mouse è scieglendo la voce New Database. A questo punto apparirà la dialog New Database che si compone di diversi gruppi di opzioni nella sezione di sinistra "Select a page". La prima pagina è General, come mostrato in Figura 2.24, che consente di definire il nome del database, il proprietario (di default è dbo), l'attivazione automatica dell'indicizzazione integrata full text e i file fisici che ospiteranno il nuovo database. Ciascun database si compone di un file di dati con estensione .MDF e uno o più file di log con estensione .LDF. In questa fase è possibile anche definire il path fisico del database. È importante sottolineare che a meno che non si disponga di soluzioni per o storage di rete centralizzato (ad esempio di sistemi SAN), è possibile salvare i file di database esclusivamente su dischi locali. In qualche caso, per ragioni prestazionali, si preferisce salvare il file dati e il file di log su dischi

PuntoInformatico Libri

diversi, in modo da sfruttare il parallelismo hardware. Ha naturalmente molto meno senso optare per partizioni diverse che risiedono fisicamente sullo stesso disco perché non si avrebbe alcun vantaggio prestazionale.

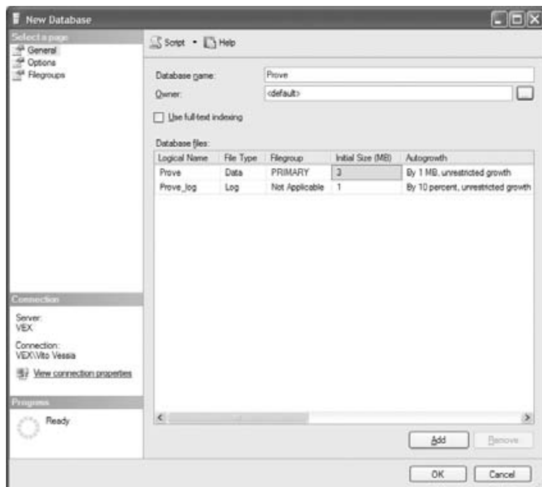


Figura 2.23: Impostazioni generali di creazione di un database

La pagina Options (Figura 2.24), invece, permette di impostare le opzioni di creazione del database. È possibile scegliere la collation, già descritta nel paragrafo relativo all'installazione di SQL Server. Di default ogni nuovo database eredita la collation di installazione di SQL Server, ma è possibile modificarla sul singolo database in modo che ogni suo oggetto, salva diversa indicazioni, erediti questa impostazione dal database e non dal motore. La seconda opzione riguarda il Recovery Model, cioè la modalità e le tecniche di recupero e di mantenimento della consistenza dei dati in caso di crash o di errori sul database, ma questo argomento avanzato non verrà trattato in questo volume e se ne rimanda alla documentazione ufficiale del

produttore. La Compatibility Level, invece, permette di impostare il livello di retrocompatibilità del database. SQL Server 2005 gestisce tre livelli di compatibilità: 90, cioè l'attuale versione 2005, 80, cioè la versione 2000 e 70, cioè la versione 7.0. Tutte le precedenti versioni non sono più supportate. Il livello di compatibilità è importante perché agisce su tutti gli aspetti del database: se impostate un livello 80, ad esempio, non potrete utilizzare nessuna delle caratteristiche nuove introdotte da SQL Server 2005 nel database: né le migliorie a T-SQL, né il supporto a .NET, né i nuovi tipi di dati, ecc... Il vostro database girare in un motore SQL Server 2005, ma vivrà a tutti gli effetti in un ambiente di emulazione di SQL Server 2005. Questa funzionalità è comoda quando si dispongono database e applicazioni basate su precedenti versioni del motore e che presenterebbero malfunzionamenti a seguito del passaggio secco alla nuova versione. Invece migrando il database ma mantenendo la compatibilità col passato è possibilità pianificare con maggior tranquillità l'upgrade ma consentendo comunque di aggiornare il motore e di sfruttare le nuove caratteristiche in database e applicazioni nuove.

Figura 2.24: Impostazioni avanzate del database

PuntoInformatico | Libri

La sezione Other Options offre alcune interessanti opzioni di configurazione del database. Osserviamone le più interessanti:

- Database ReadOnly: consente di usare il database in sola lettura;
- Restricted Access: il livello di accesso al database, può assumere i valori Multiple (sono consentiti accessi multipli al database da parte di più utenti), Single (è consentito solo un accesso, tutti quelli successivi verranno negati fino a quando l'utente esclusivo non chiude la connessione), Restricted (solo membri non livelli di autorizzazione elevati possono accedere);
- Database State: lo stato del database, esso normalmente può apparire ONLINE, cioè disponibile ed utilizzabile, OFFLINE (non accessibile, stato da impostare se si sta effettuando manutenzione sul database), RESTORING (è in corso un restore), RECOVERING (vi è un tentativo di recupero in atto a seguito di un crash di sistema o del database), RECOVERY PENDING (vi è un recupero in corso ma questa operazione si è interrotta a causa di problemi), SUSPECT (vi sono dei problemi direttamente a livello di file fisico del database), EMERGENCY (uno stato impostabile dall'utente che porta il database ad essere Readonly e in Single mode);
- Allow Shrink: i file .MDF ed .LDF, all'atto della creazione, salvo diversa indicazione, occupano uno spazio su disco ridotto e vengono incrementati in dimensione alla crescita dei dati nel database. Quando si effettuano delle cancellazioni di dati o del log, queste operazioni portano ad una minore richiesta di spazio a livello di file fisico che, pertanto, potrebbe ridursi in dimensioni; questa opzioni consente l'autoriduzione automatica. Di defualt è imposta a False per ragioni prestazionali.

Dal bottone Scripts presente nella toolbar è possibile generare i comandi T-SQL corrispondenti alla creazione di un database secondo le impostazioni del database corrente. Non si dimentichi che tutto ciò che è possibile fare con Management Studio è possibile farlo con T-SQL visto

che il primo si limita semplicemente a trasformare in comandi T-SQL le operazioni e le impostazioni gestite in modo grafico dall'utente.

2.5.3 Schemi, tabelle e campi

SQL Server 2005 introduce il concetto di schema, già presente da anni in prodotti come Oracle e DB/2, cioè una sorte di suddivisione logica degli oggetti del singolo database (tabelle, campi, viste, ecc...) in sottoinsiemi. Questa ulteriore gerarchia consente di semplificare la gestione della sicurezza in questa nuova versione del prodotto, perché è possibile associare ad un utente la visibilità direttamente a livello schema: tutti gli oggetti presenti nello schema verranno automaticamente resi disponibili a quell'utente. Tutti gli oggetti dello stesso schema fanno parte dello stesso namespace (spazio dei nomi) per cui non saranno possibili omonimie di oggetti all'intero dello stesso scema. Sarà invece possibile avere, ad esempio, due tabelle chiamate Ordini all'interno dello schema Clienti e dello schema Fornitori, perché il nome completo che assumeranno le tabelle sarano Clienti.Ordini e Fornitori.Ordini.
Per visualizzare gli schemi presenti in un database bisogna espandere il nodo Security del database e successivamente il nodo Schemas. Per creare un nuovo schema si selezionerà il nodo Schemas, e richiamato il menù contestuale col tasto destro del mouse, si invocherò la voce New Schema che farà apparire la dialog mostrata in Figura 2.25.

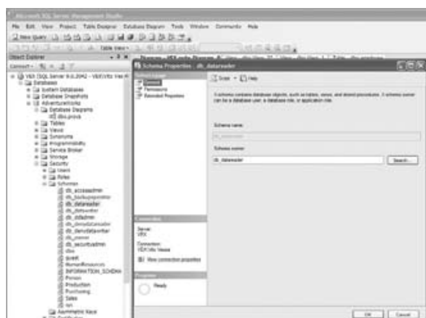


Figura 2.25: Gestione degli schemi

PuntoInformatico Libri

Dal ramo Tables del nodo database, invece, è possibile consultare l'elenco delle tabelle presenti. La creazione di una nuova tabella richiede la solita invocazione del menù contestuale, questa volta del nodo Tables, e la scelta della voce New Table. Questo comando attiverà all'interno della finestra principale Document Explorer di Management Studio, una tabella come quella mostrata in Figura 2.26 che consente di definire i campi che costituiscono la tabella.

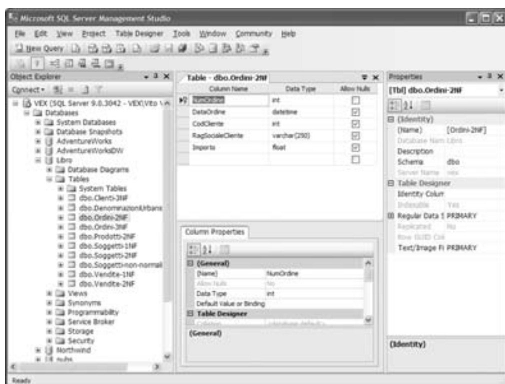


Figura 2.26: Creazione e gestione di tabelle

Per ciascun campo sarà possibile definire il nome, il tipo, selezionabile da una combo ed eventualmente la dimensione, qualora questa sia richiesta dal tipo (ad esempio CHAR e VARCHAR) e la possibilità di contenere valori null. L'impostazione dei campi che rappresentano la chiave primaria sarà facilmente realizzabile selezionando i campi che dovranno costituire la tupla e premendo l'inconfondibile chiavetta gialla stilizzata nella toolbar (Set/Remove Primary Key). Nella parte inferiore si attiverà la sezione Column Properties che riepiloga le informazione del campo selezionato e ne permette la consultazione e l'informazione di impostazioni aggiuntive (Figura 2.27) quali, ad esempio, l'attributo Identity e, di questo, il Seed (il valore di

partenza del contatore) e lo step di incremento (di default è 1).

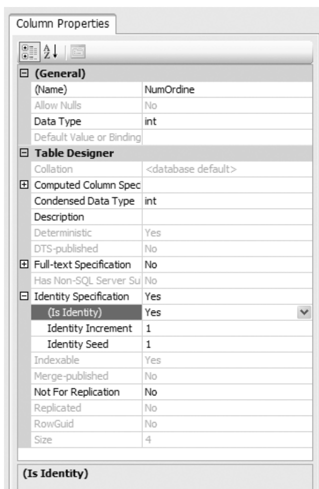


Figura 2.27: Impostazione dei campi Identity

PuntoInformatico Libri

2.5.4 Relazioni, chiavi esterne, indici e constraint

La definizione delle foreign key è un'altra importante caratteristica di SQL Server e del modello relazionale gestibile da Management Studio. Una volta avviata la creazione o la modifica di una tabella è anche possibile definirne le foreign key cliccando sul pulsante Relationships della toolbar. Questa operazione fa aprire la dialog Foreign Key Relationships (Figura 2.28) che enumera l'elenco delle foreign key presenti nella tabella e, per ciascuna di esse, il nome e le informazioni di definizione (tabella e campo di partenza e tabella e campo di arrivo). Cliccando sul bottone di lookup presente alla voce Tables and Columns Specification appaerirà un'ulteriore dialog nella quale sarà possibile modificare la foreign key specificando tabelle e campi primari differenti.

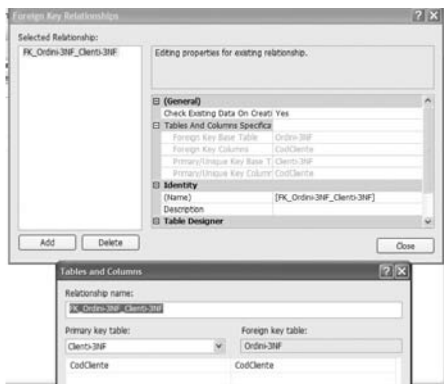


Figura 2.28: Gestione delle Foreign Key

Dalla toolbar è possibile premere anche il bottone Manage Indexes and Keys che fa apparire la dialog di gestione degli indici (Figura 2.29). Le tabelle dei database relazionali possono contenere anche milioni di righe e quando si effettua una query che recupera righe da una tabella verificando certi valori contenuti nei suoi campi, SQL Server deve leggere le righe una per una e verificare se ciascuna di esse risponde ai requisiti della query. Questa è un'attività molto onerose così, per semplificare il lavoro e accrescere conseguentemente

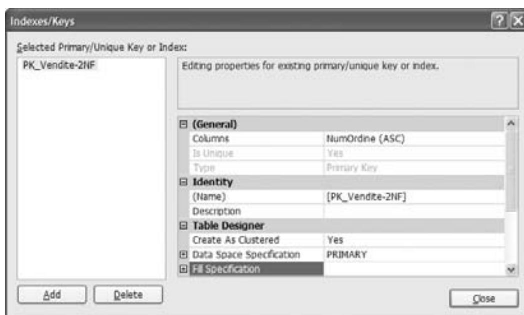


Figura 2.29: Gestione degli indici

le prestazioni, SQL Server, come tutti gli RDBMS in circolaazione, introduce il concetto di indice.
Un indice ricorda proprio l'idea dell'indice di un libro: quando si deve cercare un argomento in un libro, solitamente non lo si scorre tutto, pagina per pagina, ma si consulta l'indice che, per l'argomento ricercato, ci conduce rapidamente alla pagina in cui è trattato. In particolare, un indice SQL Server, è definito da uno o più campi di una certa tabella (ad esempio dalla coppia di campi City, Country della tabella Customers di Northwind): SQL Server costruirà una struttura intermedia, l'indice, in cui, per ogni occorrenza univoca di coppie di valori City – Country, ricorderà la posizione fisica di quelle righe della tabella che riportano la stessa coppia di valori. Così, ogni volta che in un query si farà riferimento ai valori della coppia di campi City – Country, o anche solo ad uno dei due campi (qualora non vi siano indici specifici sui singoli campi), SQL Server non effettuarà una scansione fisica della tabella (table scan) per verificare i valori di quei campi, ma consulterà il relativo indice che lo porterà direttamente alla posizione fisica dei campi della tabella che soddisfano i requisiti. E questo per tutte le condizioni testate nella query. Ove non vi siano indici, anche parziali, che rispondono all'interrogazione, SQL Server procederà con la table scan. Naturalmente è possibile che, nella stessa query, talune condizioni siano risolvibili con indici e altre con la scansione fisica. Infine, ove ad esempio la condizione di filtro preveda la verifica dei campi City, Country, PostalCode e si dispone, invece, solo di un indice City – Town, SQL Server userà l'indice per individuare tutte le righe che rispondono alla coppia City – Town ed effettuare un table scan di queste righe per verificare la terza condizione.
Ad ogni modo non si ha la possibilità diretta di intervenire su quali indici utilizzerà SQL Server. È però possibile verificare se il motore effettivamente sfrutta degli indici per una determinata query o se effettua scansioni di righe. Da questo punto di vista l'Execution Plan si dimostra uno strumento formidabile, scomponendo la query nei

PuntoInformatico Libri

vari step necessari a SQL Server per eseguire e mostrando se ci sono colli di bottiglia facilmente risolvibili con l'introduzione di nuovi indici (Figura 2.30), definibili. Tuttavia non è opportuno esagerare con la creazione di troppi indici, perché ciascun di esse occupa spazio sul database e deve costantemente essere aggiornato ad ogni inserimento, modifica o cancellazione di righe sulla tabella a cui fa riferimento, producendo un discreto dispendio di risorse.

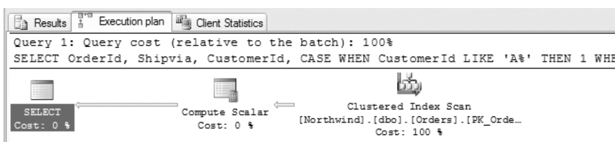


Figura 2.30: Execution Plan

Le check constraint, invece, sono delle regole che è possibile definire a livello di singolo campo. Ad esempio, se si definisce un campo Importo di tipo float, questo certamente non potrà contenere stringhe o valori più grandi della dimensione massima prevista per il tipo float, si potrà anche specificare se deve contenere null oppure no. Tuttavia queste restrizioni a volte si dimostrano insufficienti e ci sarebbe la necessità di definire veri e propri vincoli basati su regole che tengano conto dei valori di tutti i campi della riga. Ad esempio: Importo > 5000 AND TipoOrdine IN ('C', 'F')

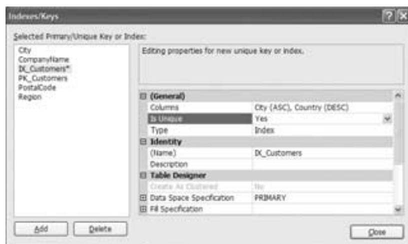


Figura 2.31: Gestione dei vincoli

Ciò è possibile cliccando il bottone Manage Check Constraints dalla solita toolbar (Figura 2.31).
Il modello relazionale è strettamente basato sulle profonde ed intricate relazioni che intercorrono tra i vari oggetti del database, dunque, ogni volta che si effettua una modifica alla sua struttura, inevitabilmente si producono effetti che vanno al di là dell'oggetto modificato. La modifica di una chiave primaria che è anche foreign key per un'altra tabella ne è il tipico esempio. Si sente l'esigenza di uno strumento che almeno indichi, per ciascun oggetto del database, le relazioni di dipendenza: cioè quali altri oggetti usa e da quali oggetti viene usato. Fortunatamente questa semplice ma efficace analisi è ora possibile con SQL Server 2005 che ha introdotto l'analisi delle dipendenze: adesso, selezionando un qualsiasi oggetto del database (tabella, vista, stored procedure, ecc...) e richiamando il menù contestuale col tasto destro del mouse, sarà quasi sempre possibile invocare la voce di menù Dependencies che, dopo una rapida analisi basata sulla consultazione delle tabelle di sistema, produrra una dialog come quella mostrata in Figura 2.32. Essa mostra l'oggetto selezionato e popola un albero di “Objects that depend on object” o di “Objects on which object depends”.

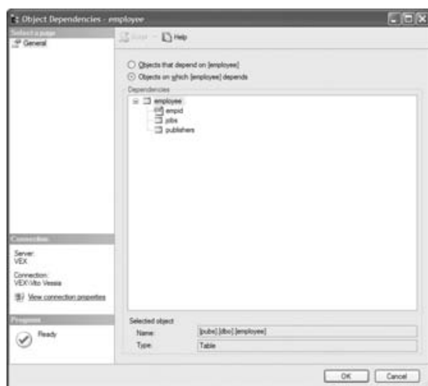


Figura 2.32: Albero delle dipendenze

2.5.5 Creazione di Viste

Le viste sono delle interrogazioni in linguaggio T-SQL che vengono memorizzate all'interno del database e che prodono resultset che possono essere fruiti come se fossero tabelle fisiche. Però il loro contenuto non viene memorizzato su disco, ma la fruzione della tabella porta sempre a rieseguire la query di base che la costituisce e quindi a leggere i dati da essa restituiti direttamente dalle tabelle fisiche di origine che compongono la query.
Tutte le viste del database sono raccolte e mostrato nel nodo Views in SQL Management Studio, sotto il nodo del database. La creazione di una vista è possibile selezionando il nodo Views, attivando il menù contestuale e selezionando la voce New View. Apparirà una dialog (Figura 2.33) che permetterà di selezionare le tabelle che saranno interessate dalla vista, elencate tra quelle presenti nel database.

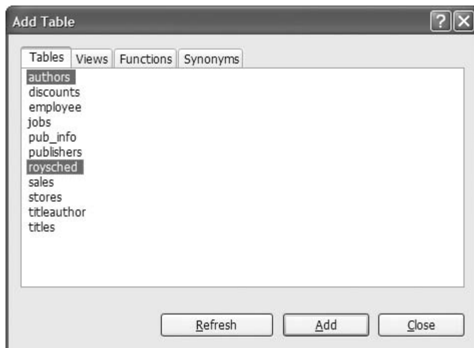


Figura 2.33: Selezione dell tabelle che comporranno la vista

A questo punto le tabelle verranno mostrate graficamente nel Document Explorer e sarà possibile comporre graficamente la query oppure scriverla e modificare direttamente in T-SQL nella finestra sottostante, analogamente a quanto si fa con il Query Editor (Figura 2.34).

Figura 2.34: Costruzione della vista

2.6 BACKUP E RESTORE

L'estrema rilevanza dei dati contenuti nei database implica delle politiche molto attente di conservazione di copie di sicurezza dei dati e così, le fasi di backup e restore di un database relazionale assumano un'importanza decisiva. Le cause di perdita dei dati possono essere molteplici ma possiamo classificarle nei seguenti principali gruppi:

- errori di programma;
- errori di amministrazione;
- errori di sistema (memoria, disco, ecc...);
- incidenti esterni (incendi o altri eventi gravi).

Gli errori di programma possono essere gestiti facilmente con il supporto alle transazioni offerto da SQL Server. Ove questo non sia possibile o il problema dipenda dalle altre tre cause, il backup e la corrisponde operazione di restore restano l'unico modo per riportare i dati in una situazione consistente senza perdere troppo lavoro. SQL Server offre quattro modalità di backup:

- backup completo;

- backup differenziale;
- backup di log di transazioni;
- backup di file e di filegroup del database.

Il backup completo è un'operazione che preserva l'immagine completa del database al momento dell'operazione di backup e quindi i dati, lo schema e lo struttura di file del database. Il backup differenziale rappresenta un step successivo rispetto ad un database completo e consente di modificare le differenze (di dati e schema) intervenute dall'ultimo database completo. Pertanto il suo ripristino richiede obbligatoriamente la presenza anche del database completo. Pur dipendendo da un database completo, aspetto che non rende questa soluzione autoconsistente, il database differenziale ha l'indubbio vantaggio di essere rapido da produrre e di richiedere non molte risorse. Naturalmente la sua dimensione dipende dal tempo e dalla quantità di modifiche intervenute rispetto al database completo a cui fa riferimento. Un backup di log di transazione è anch'esso una forma di database differenziale basato sul tracciamento di tutte le operazioni di scrittura effettuate sul database dall'ultimo backup completo. In seguito verrà approfondita la problematica relativa alla gestione delle transazioni. Dunque, semplicemente riapplicando nella giusta sequenza cronologica tutte queste operazioni (INSERT, UPDATE e DELETE) sul database ripristinato dal backup completo, si è in grado di ritornare alla situazione del database al momento del backup del log transazionale. Infine, sappiamo che ciascun database è costituito da due o più file fisici .MDF ed .LDF. SQL Server 2005 consente di effettuare il backup e il successivo ripristino di singoli file fisici che costituiscono il database in luogo dell'intero database.

l Il backup da SQL Management Studio

Per effettuare il backup di di un database da Management Studio, è sufficiente selezionare il nodo corrisponde al nome del database da Object Explorer, invocarne il menù contestuale col tasto destro del mouse e scegliere Tasks --> Backup. A questo punto appare una dialog

Back Up Database come mostrato in Figura 2.35.

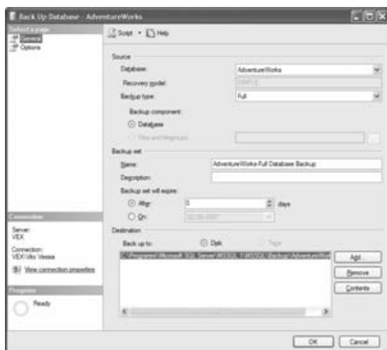


Figura 2.35: Backup di un database

Essa, come nello standard delle dialog di popup di Management Studio, si compone di una serie di pagine. La prima è General che permette di selezionare il database di cui effettuare il backup (di default viene proposto il database da cui è partita la richiesta), la tipologia di backup (Full, completo o Differential, differenziale), il nome descritto che verrà associato al backup, l'eventuale scadenza, se

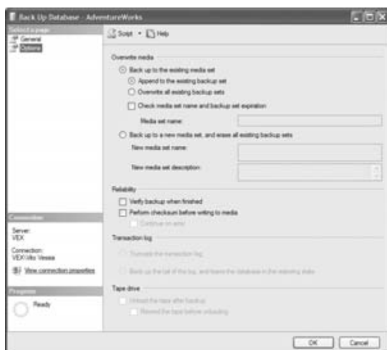


Figura 2.36: Opzioni aggiuntive sul Backup di un database

PuntoInformatico Libri

c'è (è possibile impostare questa proprietà per evitare di ripristinare backup troppo vecchi) e il file fisico su cui effettuare il backup. La seconda pagina Options, invece, come mostrato in Figura 2.36, specifica ulteriori informazioni.
L'opzione Overwrite Media consente se utilizzare un file fisico di backup già esistente, che per convenzione dovrebbe avere un'estensione .BAK, oppure crearne uno nuovo. Nel primo caso è possibile aggiungere il nuovo backup nel set esistente oppure eliminare tutti i backup già presenti nel file e creare un nuovo. Infatti ciascun file di backup può ospitare al suo interno più set di backup, ognuno contraddistinto da un nome, una descrizione e una data.

2.6.1 Restore del database da SQL Management Studio

L'operazione di restore è sempre effettuabile dallo strumento di amministrazione centralizzato di SQL Server 2005. È sufficiente selezionare il nodo corrisponde al nome del database da Object Explorer, invocarne il menù contestuale col tasto destro del mouse e scegliere Tasks --> Restore --> Database (oppure Files and File Group). Ci concentreremo esclusivamente sulla prima opzione, rimandando il restore dei File e dei File Group alla documentazione ufficiale). A

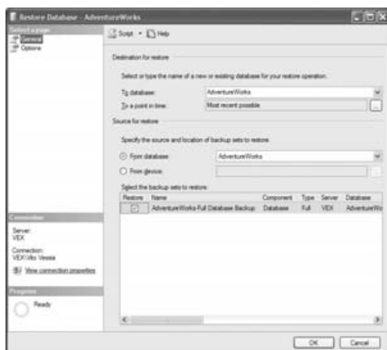


Figura 2.37: Opzioni aggiuntive sul Backup di un database

questo punto appare una dialog Back Up Database come mostrato in Figura 2.37.
La pagine Options descrive il nome del database su cui effettuare il restore. Scegliendo il nome tra quelli non già esistenti sul database engine, sarà possibile creare al volo un nuovo database a partire da restore. Nella sezione inferiore Select the backup set to restore vengono riepilogati i backup set contenuti nel file di backup. È possibile selezionarne uno solo per il restore. La pagina successiva Options (Figura 2.40) scende in dettaglio con la configurazione di ripristino.

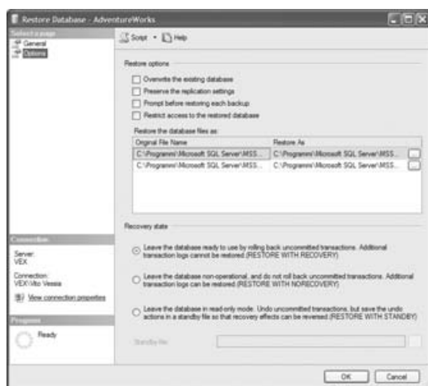


Figura 2.38: Opzioni aggiuntive sul Backup di un database

L'opzione Overwrite over existing database consente di sovrascrive il contenuto esistente qualora si sia scelto come database di arrivo del restore uno già esistente. La sezione Restore the database file as è la più rognosa di tutta la fase di restore perché prevede l'indicazione precisa di e path nome dei file fisici e del database. Esso propone automaticamente i file e i path del database sul database engine di partenza. A questo punto bisogna modificare queste impostazioni per renderle compatibili con la configurazione di dischi e partizioni del database engine di arrivo.

2.7 ATTACH E DETACH

SQL Server offre la comoda possibilità di agganciare a caldo i file .MDF ed .LDF, dunque i file fisici, di un database preesistente, magari persino proveniente da un altro database engine installato su un'altra istanza o su un'altra macchina. L'unico requisito è che tale database non sia attualmente agganciato al database engine di provenienza, che ne imposterebbe un lock alla Win32, e che il database engine di arrivo sia alla stessa versione di quello di partenza (prereferibilmente persino a livello di Service Pack installati). Il database così agganciati conservare tutti i dati, gli schemi, gli utenti e le impostazioni originarie (compresa la collation) anche se il database engine di arrivo fosse configurato con impostazioni differenti. Per effettuare l'attach da SQL Management Studio è sufficiente selezionare il nodo Databases da Object Explorer, richiamare il menù contestuale e scegliere la voce Attach. Questo comando farà apparirà la dialog mostrata in Figura 2.39.

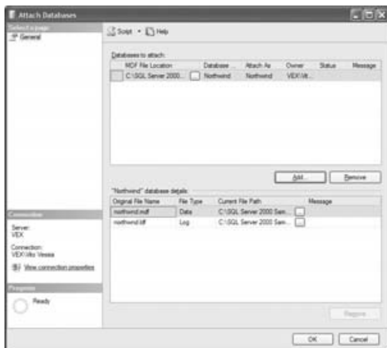


Figura 2.39: Opzioni aggiuntive sul Backup di un database

Essa presenta la sola pagina General dalla quale è possibile selezionare il file .MDF del database di cui effettuare l'attach. Una volta effettuata questa scelta, la sezione inferiore Database Details riepilo-

ga le informazioni associate al database relativamente ai suoi file fisici. È infatti sufficiente selezionare il file .MDF per recuperare anche le informazioni relative al log transazionale. Scegliendo OK, il database viene agganciato al nuovo database engine divenendo parte a tutti gli effetti, alla stregua di tutti gli altri database presenti. È necessario che il file .MDF sia fisicamente presente, e che dunque sia stato copiato, su un disco fisico del database engine di arrivo.

Per effettuare il detach di un database da Management Studio, è sufficiente selezionare il nodo corrisponde al nome del database da Object Explorer, invocarne il menù contestuale col tasto destro del mouse e scegliere Tasks --> Detach. La dialog mostrata in Figura 2.40 apparirà.

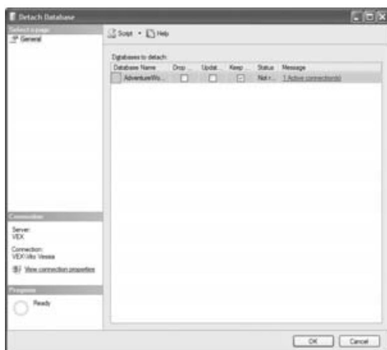


Figura 2.40: Opzioni aggiuntive sul Backup di un database

La dialog consente semplicemente di riconfermare il file di cui effettuare il detach. Confermando l'operazione con l'OK, il database verrà sganciato dal database engine, dal quale non sarà più utilizzabile, e potrà essere copiato e trasportato su altri database engine (o sullo stesso in seguito) per un nuovo attach.

PuntoInformatico Libri

IL LINGUAGGIO TRANSACT-SQL

Il linguaggio Transact SQL (o T-SQL) è il dialetto proprietario SQL di Microsoft SQL Server 2005. È però largamente compatibile con lo standard SQL. Pertanto, conoscere bene T-SQL consente di passare ad altri dialetti SQL altrettanto avanzati con grande facilità. La versione 2005 di T-SQL introduce una serie di novità che vedremo nel corso di questo capitolo.

3.1 INTRODUZIONE AL LINGUAGGIO SQL

SQL non identifica un prodotto commerciale, ma un linguaggio di definizione e di interrogazione dei dati, per cui è logicamente vicino ai linguaggi di programmazione, anche se con scopi ben più circoscritti. Analogamente a quanto avviene per linguaggi come Pascal e C, il suo nome non indica un prodotto commerciale, bensì un modello sintattico e teorico generico.
Il suo scopo è eseguire varie operazioni sia sui dati che sulle strutture che li contengono. La sigla, acronimo di Structured Query Language, è ormai diventata sinonimo di linguaggio standard per la gestione dei database relazionali. L'acronimo si traduce letteralmente come Linguaggio di interrogazione strutturato.
SQL assolve alle funzioni di Data Description Language (acronomimo DDL, linguaggio di descrizione dei dati e delle strutture che li conterranno), di Data Manipulation Language (acronimo DML, linguaggio per la manipolazione dei dati) e di linguaggio di interrogazione.
La potenza e la relativa facilità d'uso di questo linguaggio, almeno fino a quando non si intende procedere ad interrogazioni particolarmente complesse, ne consente l'uso non solo ad informatici di professioni o, peggio, ad esperti di database relazionali, ma anche ad utenti di computer normali che intendano interrogare i propri database in modo anche molto approfondito. La sua relativa linearità, poi, ha consentito negli ultimi anni di far nascere un folto gruppo di strumenti software grafici che permettono di disegnare graficamen-

PuntoInformatico | Libri

te la query, senza scrivere una sola parola della sintassi SQL. Saranno poi questi strumenti a generare la query SQL per l'utente in base a quanto disegnato dall'utente. Microsoft Access ne è un esempio su tutti. In realtà questi strumenti RAD SQL sono ormai così potenti e precisi, che vengono adoperati anche dai professionisti informatici (sviluppatori di applicazioni sui database) per generare almeno la prima versione delle loro query che raffineranno successivamente. SQL Server Management Studio e il modulo di query di Visual Studio 2005 ne sono solo due esempi. Questi strumenti sono così avanzati da consentire addirittura quello che si definisce reverse engineering, cioè di rigenerare il modello grafico di una query a partire dal codice SQL della query stessa.
Non bisogna, però farsi ingannare, perché se da un lato SQL è intuitivo e semplice, da un altro, per essere capito a fondo richiede di essere studiato con attenzione per capirne le notevoli potenzialità.

3.1.1 Storia di SQL

La storia di SQL, che si pronuncia facendo lo spelling inglese delle lettere che lo compongono, e quindi ess-chiu-el e non "siquel" come si sente spesso, inizia nel 1974 con la definizione da parte di Donald Chamberlin e di altre persone che lavoravano presso i laboratori di ricerca dell'IBM di un linguaggio per la specificazione delle caratteristiche dei database che adottavano il modello relazionale. Questo linguaggio si chiamava SEQUEL (Structured English Query Language) e venne implementato in un prototipo chiamato SEQUEL-XRM fra il 1974 e il 1975. Le sperimentazioni con tale prototipo portarono fra il 1976 ed il 1977 ad una revisione del linguaggio (SEQUEL/2), che in seguito cambiò nome per motivi legali, diventando SQL. Nonostante questa variazione, molti hanno continuato a pronunciare SQL come sequel. Il prototipo (System R) basato su questo linguaggio venne adottato ed utilizzato internamente da IBM e da alcuni sui clienti scelti. Grazie al successo di questo sistema, che non era ancora commercializzato, anche altre compagnie iniziarono a sviluppare

i loro prodotti relazionali basati su SQL. A partire dal 1981 IBM cominciò a rilasciare i suoi prodotti relazionali e nel 1983 cominciò a vendere DB2. Nel corso degli anni ottanta numerose compagnie (Oracle, Sybase e altre) commercializzarono prodotti basati su SQL che divenne lo standard industriale di fatto per quanto riguarda i database relazionali.
Nel 1986 l'ANSI adottò SQL (sostanzialmente si trattava del dialetto SQL di IBM) come standard per i linguaggi relazionali e nel 1987 esso diventò anche standard ISO. Questa versione dello standard va sotto il nome di SQL/86. Negli anni successivi esso ha subito varie revisioni che hanno portato prima alla versione SQL/89 e successivamente alla attuale SQL/92. La specifica continua ad evolversi anche se con maggiore lentezza.
Il fatto di avere uno standard definito per un linguaggio per database relazionali, apre potenzialmente la strada alla intercomunicabilità fra tutti i prodotti che si basano su di esso. Dal punto di vista pratico purtroppo le cose andarono differentemente. Infatti in generale ogni produttore adotta ed implementa nel proprio database solo il cuore del linguaggio SQL (il cosiddetto Entry level o al massimo l'Intermediate level), estendendolo in maniera proprietaria a seconda della propria visione del mondo dei database.
Attualmente è in corso un processo di revisione del linguaggio da parte dei comitati ANSI e ISO, che dovrebbe portare alla definizione di ciò che al momento è noto come SQL3. Le caratteristiche principali di questa nuova incarnazione di SQL dovrebbero essere la sua trasformazione in un linguaggio stand-alone (mentre ora viene usato come linguaggio ospitato in altri linguaggi) e l'introduzione di nuovi tipi di dato più complessi per permettere, ad esempio, il trattamento di dati multimediali.

3.2 LO STATEMENT SELECT

SELECT è davvero il cuore di SQL. Senza questa istruzione, l'SQL sarebbe non troppo dissimile da altri linguaggi di programmazione che

PuntoInformatico | Libri

consentono di creare strutture dati dinamiche e di aggiungere, rimuovere e modificare dati a queste strutture. Quello che rende SQL davvero unico e che dà un senso al termine Query del suo acronimo è proprio questa istruzione che semplicemente consente recuperare i dati del database attraverso delle interrogazioni. I programmatori di software gestionali sanno quanto sia importante scrivere la giusta query e quanto tempo, della loro esistenza professonale, che poi è parte dell'esistenza generale di ciascun programmatore, nella definizione di intricate SELECT che, in un sol colpo, consentano di recuperare un intero set, anche molto complesso, di informazioni. Nulla del genere, allo stato attuale, è ancora in grado di fare altrettanto scrivendo esclusivamente codice di programmazione.
Dunque, indipendemente se sarà l'utente finale a scrivere la propria query, o sarà un query costruita attraverso una maschera di programma (QBE) o sarà fissa all'interno di un'applicazione, quella che verrà eseguita sul database sarà sempre un query SELECT. Essa si compone di alcune parti convenzionalmente presenti e che fanno uso di parole chiavi riservate. Queste parti si definiscono clausole. Esserviamone la struttura:

```
SELECT    nome_colonna1 [, nome_colonna2 [, nome_colonnaN]]
FROM      nome_tabella1 [, nome_ tabella2 [, nome_tabellaN]]
WHERE     <condizione_di_ricerca>
GROUP BY nome_colonna1 [, nome_colonna2 [, nome_colonnaN]]
HAVING    <condizione_di_ricerca>
```

Si possono riconoscere le diverse clausole:

- SELECT: è obbligatoria; viene usata per specificare le colonne il cui valore si intende recuperare con l'interrogazione; questi campi possono essere tratti da tabelle, da viste, da subquery o possono essere campi calcolati come risultati di espressioni aritmetiche (es. prezzo_unitario * quantità);

- FROM: è obbligatoria ed indicate le tabelle (o le viste o le subquery) da cui sono tratti i campi dell'interrogazione;
- WHERE: è fondamentale ma non obbligatoria; indica una condizione di filtro da operare sulle righe estratte dell'interrogazione; è fondamentale osservare che lo scopo di questa clausola è di applicare a ciascuna riga restituita dall'interrogazione la condizione di ricerca: se il risultato del confronto per la riga è true, essa verrà mantenuta, diversamente essa verrà scartata; la condizione di ricerca o di filtro viene identificata anche con il termine di Predicato;
- GROUP BY: è una clausola opzionale; consente di raggruppare righe con valori uguali in modo da evitare risultati duplicati o ridondanti nell'interrogazione;
- HAVING: è una clausola opzionale che va usata solo se è presente anche la clausola GROUP BY; di questa, infatti, ne rappresenta la condizione di filtro, una sorta di WHERE relativa alla sola parte GROUP BY.

3.2.1 Le clausole obbligatorie SELECT e FROM

Facendo rifermento al database di esempio Northwind, osserviamo la seguente query:

```
SELECT *
FROM Orders
```

In essa possiamo osservare la presenza delle sole due clausole obbligatorie SELECT e FROM. Nella prima c'è la parola chiave jolly * che consente di recuperare tutti i campi di tutte gli oggetti descritti nella clasola FROM che, nell'esempio, contiene la sola tabella Orders (Figura 3.1).
È quasi sempre sconsigliata la sintassi * perché produce ambiguità e tende a far restituire sempre informazioni superflue (campi non

Table - dbo.Orders | VEX.Northwind - SQLQuery4.sql*

Column Name	Data Type	Allow Nulls
OrderID	int	☐
CustomerID	nchar(5)	☑
EmployeeID	int	☑
OrderDate	datetime	☑
RequiredDate	datetime	☑
ShippedDate	datetime	☑
ShipVia	int	☑
Freight	money	☑
ShipName	nvarchar(40)	☑
ShipAddress	nvarchar(60)	☑
ShipCity	nvarchar(15)	☑
ShipRegion	nvarchar(15)	☑
ShipPostalCode	nvarchar(10)	☑
ShipCountry	nvarchar(15)	☑
		☐

Figura 3.1: Struttura della tabella Orders

necessari) e a consumare risorse di sistema inutilmente (risorse di calcolo di SQL Server, memoria per ospitare i risultati, banda di rete occupata per trasferirli). Pertanto è preferisce sempre indicare puntualmente i campi:

```
SELECT OrderId, CustomerId, EmployeeId, ShipName FROM Orders
```

In Figura 3.2 possiamo osservare il risultato della query eseguita in SQL Server Management Studio, con indicazione puntuale dei quattro campi da restituire.
È interessante osservare la possibilità di indicare i nomi dei campi con alias e cioè in maniera diversa dal nome fisico del campo nella tabella. È possibile farlo in due modi. Si osservi la seguente query:

```
SELECT OrderId as CodiceOrdine, CustomerId, EmployeeId, ShipName
                                        DescrizioneSpedizione
FROM Orders
```

VEX.Northwind - SQLQuery4.sql* | VEX.AdventureW... SQLQuery3.sql | Table

```
SELECT OrderId, CustomerId, EmployeeId, ShipName
FROM Orders
```

Results | Messages

	OrderId	CustomerId	EmployeeId	ShipName
1	10248	VINET	5	Vins et alcools Chevalier
2	10249	TOMSP	6	Toms Spezialitäten
3	10250	HANAR	4	Hanari Carnes
4	10251	VICTE	3	Victuailles en stock
5	10252	SUPRD	4	Suprêmes délices
6	10253	HANAR	3	Hanari Carnes
7	10254	CHOPS	5	Chop-suey Chinese
8	10255	RICSU	9	Richter Supermarkt
9	10256	WELLI	3	Wellington Importadora
10	10257	HILAA	4	HILARION-Abastos
11	10258	ERNSH	1	Ernst Handel
12	10259	CENTC	4	Centro comercial Moctezuma
13	10260	OTTIK	4	Ottilies Käseladen
14	10261	QUEDE	4	Que Delícia
15	10262	RATTC	8	Rattlesnake Canyon Grocery
16	10263	ERNSH	9	Ernst Handel

Figura 3.2: La prima query

Rispetto alla query precedente, CustomerId e EmployeeId non hanno subito variazioni. Invece il campo chiave primaria OrderId è indicato nella forma OrderId as CodiceOrdine, che nel risultato della query diventerà il nome del campo in sostituzione di OrderId. L'altra forma per utilizzare gli alias è mostrata per il campo ShipName a cui abbiamo attribuito l'alias DescrizioneSpedizione. Infatti la parola chiave as può essere omessa. È altresì possibile associare un alias anche alle tabelle o agli oggetti indicati nella clausola FROM:

```
SELECT OrderId as CodiceOrdine, CustomerId, EmployeeId, ShipName
                                                DescrizioneSpedizione
FROM Orders as MieiOrdini
```

I nomi di campi, tabelle, viste, stored procedure e di ogni altro oggetti definibile nel database possono anche essere racchiusi tra parentesi quadre. Questa opzione non è necessaria quando il nome dell'oggetto non presenta spazi, possibilità consentita da SQL Server, ma lo diventa in quest'ultimo caso. Osserviamo il seguente esempio:

PuntoInformatico | Libri

```
SELECT * FROM [Order Details]
```

La tabella Order Details presenta uno spazio nel nome. Se omettessimo le parentesi quadre, la query diventerebbe:

```
SELECT * FROM Order Details
```

SQL Server considererebbe Order il nome della tabella e Details il suo alias da usare nella query e dunque ci segnalerebbe il seguente errore visto che la tabella Order non esiste:

```
Msg 156, Level 15, State 1, Line 2
Incorrect syntax near the keyword 'Order'.
```

Un'altra interessante possibilità è offerta dai campi calcolati. È infatti possibile inserire come nomi dei campi intere espressioni aritmentiche, booleane o che fanno uso di stringhe, che verranno calcolate e risolte al volo durante lo svolgimento della query. Nell'espressione tipicamente possono esserci valori costanti o anche altri campi estraibili dalla query. Naturalmente il valore di questi campi può variare da riga a riga, ragion per cui SQL Server è costretto a ricacolare l'espressione ad ogni nuova riga. Osserviamo il seguente esempio:

```
SELECT OrderId, Freight, (Freight * 0,7328) as EuroFreight
FROM Orders
```

Il campo Freight (costo di trasporto), di tipo Money, è espresso in dollari. Se però vogliamo convertire il suo valore in euro, possiamo effettuare la conversione direttamente nella query moltiplicandolo per il suo tasso di cambio odierno. Al nuovo campo calcolato così ottenuto abbiamo attribuito l'alias EuroFreight. In Figura 3.3 ne possiamo osservare il risultato.
Ma non è finita qui: è possibile applicare alle espressioni calcolate

VEX.Northwind - SQLQuery5.sql* | Table - dbo.Orders | VEX.Northwind - SQLQ

```
SELECT OrderId, Freight, Freight * 0.7328 as EuroFreight
FROM Orders
```

Results | Messages

	OrderId	Freight	EuroFreight
1	10248	32,38	23.72806400
2	10249	11,61	8.50780800
3	10250	65,83	48.24022400
4	10251	41,34	30.29395200
5	10252	51,30	37.59264000
6	10253	58,17	42.62697600
7	10254	22,98	16.83974400
8	10255	148,33	108.69622400
9	10256	13,97	10.23721600
10	10257	81,91	60.02364800
11	10258	140,51	102.96572800
12	10259	3,25	2.38160000
13	10260	55,09	40.36995200
14	10261	3,05	2.23504000

Figura 3.3: Alias e campi calcolati

anche funzioni di sistema, funzioni utente (che vedremo nel capitolo successivo) e cast. Osserviamo la seguente variante della query precedente:

```
SELECT OrderId, cast(round(Freight * 0.7328, 2) as float) as EuroFreight
FROM Orders
```

In questo caso abbiamo applicato numerose trasformazioni sul campo calcolato originario. Innanzitutto è stata applicata la funziona Round, una funzione nativa di SQL Server che arrotando un numero (il primo argomento della funzionae) con una precisione di un numero di cifre decimali espressa dal secondo argomento. Nel caso specifico abbiamo arrotondato a 2 cifre decimali. Infine, visto che il numero si presentava 23.73000000, cioè con una serie di inutili zeri dopo i due decimali, abbiamo applicato l'istruzione CAST di SQL Server che effettua il cast di un tipo ad un altro, analogamente a quanto si fa con i linguaggi di programmazione tradizionale. Nell'esempio il CAST ci consente di ricondurre il numero ad un float, come si può osservare in Figura 3.4.

PuntoInformatico Libri

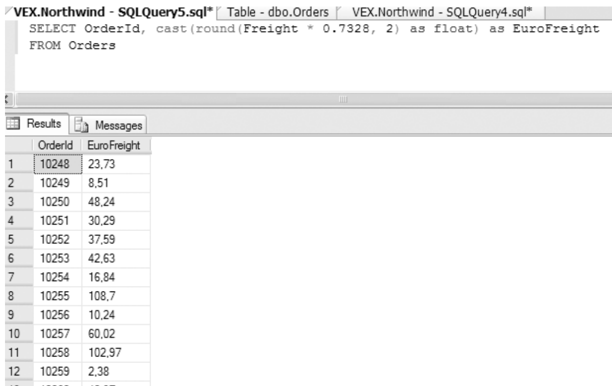


Figura 3.4: Alias e campi calcolati, versione finale

A volte il valore restituito dalla SELECT non è sufficiente come risultato finale, ma su di esso vanno applicate delle trasformazioni o degli ulteriori controlli. Ad esempio, in presenza di un NULL si vuole restituire un valore convenzionale oppure una singola informazione atomica è spalmata su due campi della query che sono mutuamente esclusivi (se è valorizzato uno dei due, l'altro deve essere NULL, in questo caso non ha senso restituire due campi ma ne basta uno solo nella query che contenga il valore non NULL). Della CASE esistono due sintassi. Una semplificata:

```
CASE espressione_1
    <WHEN espressione_1 THEN risultato_1>
    <WHEN espressione_2 THEN risultato_n>
    [ELSE risultato_m]
END
Osserviamo un esempio:
SELECT OrderId, Shipvia,
CASE Shipvia
    WHEN 1 THEN 10
```

```
    WHEN 2 THEN 20
    ELSE 0
END ShipViaSurrogato
FROM Orders
```

Si osservi come su sul campo Shipvia venga applicata una CASE che opera una vera e propria trasformazione di valori. Se il valore presente nel campo vale 1 allora come risultato della CASE si avrà un 10, se è 2 si avrà un 20, in tutti gli altri casi si avrà uno 0. Al risultato del campo così calcolato viene attributi l'alias ShipViaSurrogato. Questa versione semplificata consente di effettuare valutazioni solo sul valore del campo da trasformare. La Figura 3.4.1 mostra un esempio di questa query.

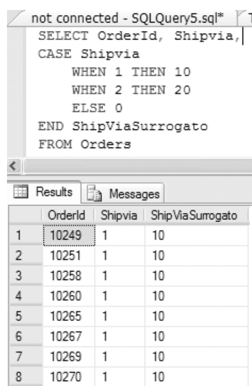


Figura 3.5: L'istruzione CASE semplice

Ne esiste anche una versione più potente, detta con ricerca, che consente di estendere le valutazioni anche ad altri campi e con espressioni condizionali ben più complesse. Praticamente si può fare quasi tutto quello che si fa nella clausola WHERE, che vedremo a seguire, ma direttamente a livello di clausola SELECT. Eccone la sintassi:

PuntoInformatico Libri

```
CASE
    <WHEN espressione_1 THEN risultato_1>
    <WHEN espressione_2 THEN risultato_n>
    [ELSE risultato_m]
END
Osserviamo la seguente query di esempio:
SELECT OrderId, Shipvia, CustomerId,
CASE
    WHEN CustomerId LIKE 'C%' THEN 1
    WHEN ShipVia IN (1, 2) AND OrderId > 10400 THEN 20
    ELSE ShipVia
END ShipViaSurrogato
FROM Orders
```

Anche in questo caso l'esempio ha poco senso da un punto di vista logico ma risulta efficace per mostrare la potenza del costrutto. Questa seconda CASE di esempio, infatti, va interpretata così: innanzitutto, in questa versione con ricerca si perde il concetto di espressione di riferimento su cui semplicemente valutare il risultato nelle parti WHEN, ma ciascuna parte WHEN può avere al suo interno una espressione condizionale completamente diversa e anche piuttosto complessa. Infatti nel primo WHEN abbiamo una CustomerId LIKE 'C%', nel secondo ben due condizioni in AND tra loro (ShipVia IN (1, 2) AND OrderId > 10400). L'immancabile ELSE consente di chiudere il cerchio.
Si consideri che entrambe le versioni della CASE agiscono in corto circuito: cioè non appena viene verificata come vera una condizione, viene attribuito come risultato finale della case il suo valore THEN e non vengono eseguite le altre WHEN. Pertanto è importante ricordarsi di scrivere le WHEN in ordine di priorità.
Un'altra potente possibilità è offerta dall'opzione TOP inseribile nella clausola SELECT. Essa consente di restituire un numero massimo o una percentuale massima di righe nel resultset. Eccone la sintassi

completa nella clausola SELECT:

```
SELECT [TOP n [PERCENT] [WITH TIES]] nome_colonna1 [, nome_colonnaN]
```

L'argomento n rappresenta il numero massimo di righe da restituire; se abbinato con il parametro PERCENT, n deve essere interpretato come la percentuale di righe da restituire rispetto al totale di righe che la query restituirebbe in assenza della TOP. Eccone un esempio:

```
SELECT TOP 11 OrderId, CustomerId, Freight FROM Orders
E un altro con la PERCENT:
SELECT TOP 20 PERCENT OrderId, CustomerId, Freight FROM Orders
```

Infine si può l'opzione WITH TIES offre un'interessante funzionalità: a volte, quando intendiamo recuperare solo una parte di righe rispetto al totale ritornato dalla query, non possiamo semplicemente indicare un numero o una percentuale secca, ma si pone l'esigenza di recuperare questo blocco di righe per rottura di codice rispetto ad uno dei campi. Osserviamo un esempio derivato dalle due query dimostrative precedenti:

```
SELECT TOP 11 WITH TIES OrderId, CustomerId, Freight
FROM Orders
ORDER BY CustomerId
```

Nella query abbiamo dato, ma questa volta solo come indicazione non tassativa, l'ordine di recuperare le prime 11 righe. Però, ordinando le righe per il campo CustomerId, la WITH TIES impone che il flusso di righe da restituire non debba essere interrotto esattamente alla riga undicesima, ma che prendendo come riferimento il campo CustomerId, indicato nella ORDER BY (cluasola che esamineremo nel corso del capitolo), il flusso verrà interrotto solo quando tutte eventuali righe successive alla undicesima abbino un CustomerId differente.

PuntoInformatico | Libri

Altre due interessanti parole chiave utilizzabili nella clausola SELECT sono CAST e CONVERT: esse permettono di convertire un tipo di valore in un altro, applicando anche trasformazioni complesse durante la conversione. Ecco la sintassi della CAST:

```
CAST ( espressione AS tipo_dato [ (lunghezza ) ])
```

Un esempio:

```
SELECT CAST(OrderId AS VARCHAR(20)),  FROM Orders
```

I tipi di partenza e di arrivo sono quelli previsti da SQL Server. È evidente che il cast, per sua stessa natura, non è sempre possibile. Sarà sempre possibile convertire un numero (intero o decimale) o una data in una stringa, ma non è detto che si possa fare il contrario. Osserviamo questo esempio:

```
SELECT CAST('34A1' AS INT) FROM Orders
```

Stiamo cercando di convertire una stringa alfanumerica in un numero. SQL Server ci segnalerà il seguente errore:

```
Msg 245, Level 16, State 1, Line 1
Conversion failed when converting the varchar value '34A1' to data type
int.
```

La CONVERT, invece, è ben più sofisticata. Eccone la sintassi:

```
CONVERT ( tipo_dato [ ( lunghezza ) ] , espressione [ , stile ] )
```

A parte una diversa posizione degli argomenti, in CONVERT ritroviamo l'argomento aggiuntivo stile. Esso indica delle regole aggiuntive di conversione. In effetti la CONVERT va oltre il semplice cast, ma

effettua vere e proprie trasformazioni di valori. A causa del ridotto spazio a disposizione in questo volume, si rimanda alla consultazione della documentazione ufficiale di SQL Server.

3.2.2 La clausola WHERE

Come già accennato, la clausola WHERE specifica un'espressione booleana che viene valutata per ogni riga risultante dalla query. Qualora la valutazione sia true, la riga viene mantenuta, diversamente viene tagliata. Pertanto questa clausola è il motore di filtro dell'istruzione SELECT. Osserviamo un esempio semplice di WHERE, sempre facendo riferimento alla nostra tabella Orders:

```
SELECT OrderId, Freight, CustomerId, ShipName
FROM Orders
WHERE OrderId = 10251
```

In questo caso l'espressione da valutare è OrderId = 10251. Tutte le righe che soddisferanno questa espressione verranno mantenute, le altre verranno scartate. Con tutta evidenza, considerando che OrderId è chiave primaria, avremo come possibile risultato 0 o 1 sola riga, data l'univocità di valori della chiave primaria. La Figura 3.5 ci mostra il risultato.

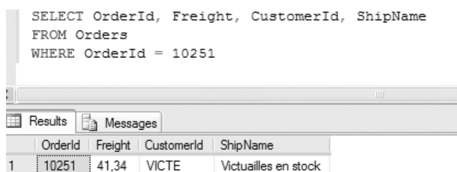


Figura 3.6: Una semplice clausola WHERE

Si possono naturalmente esprimere condizioni meno filtranti che non agiscono su valori secchi ma su range:

PuntoInformatico | Libri

```
SELECT OrderId, Freight, CustomerId, ShipName
FROM Orders
WHERE Freight >= 550
```

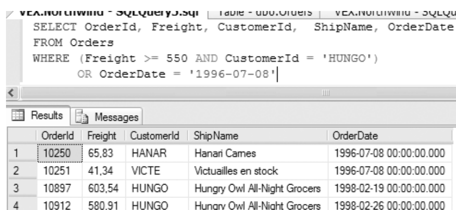
In questo caso la valutazione è meno stringente. Gli operatori di confronto semplici utilizzabili sono:

- = (uguale);
- <> (diverso);
- > (maggiore);
- < (minore);
- >= (maggiore o uguale);
- <= (minore o uguale).

Inoltre è possibile combinare tra loro più espressioni di confronto nella stessa clausola WHERE usando gli operatori booleani:

```
SELECT OrderId, Freight, CustomerId,  ShipName, OrderDate
FROM Orders
WHERE (Freight >= 550 AND CustomerId = 'HUNGO')
    OR OrderDate = '1996-07-08'
```

Nell'esempio compaiano ben tre espressioni di valutazione: una su un valore numerico (Freight >= 550), una su una stringa (CustomerId = 'HUNGO') e l'ultima su una data. Le prime due espressioni so-



	OrderId	Freight	CustomerId	ShipName	OrderDate
1	10250	65,83	HANAR	Hanari Carnes	1996-07-08 00:00:00.000
2	10251	41,34	VICTE	Victuailles en stock	1996-07-08 00:00:00.000
3	10897	603,54	HUNGO	Hungry Owl All-Night Grocers	1998-02-19 00:00:00.000
4	10912	580,91	HUNGO	Hungry Owl All-Night Grocers	1998-02-26 00:00:00.000

Figura 3.7: L'uso degli opetatori booleani nella WHERE

no in AND tra loro e sono messe in parentesi. Il tutto è in OR con l'ultima espressione. Naturalmente valgono le regole booleane di precedenza degli operatori.
È possibile utilizzare anche l'operatore booleano di negazione, il NOT, che inverte il valore logico dell'espressione che si va a valutare:

```
SELECT OrderId, Freight, CustomerId,  ShipName, OrderDate
FROM Orders
WHERE (Freight >= 550 AND CustomerId = 'HUNGO')
OR NOT (OrderDate <> '1996-07-08')
```

Abbiamo praticamente riscritto la query precedente introducendo il NOT e sostituendo l'espressione (OrderDate = '1996-07-08') con NOT (OrderDate <> '1996-07-08'). Si introduce, così, una doppia negazione che, nella logica booleana, si trasforma in un'affermazione. Infatti la nuova query restituisce un risultato identico alla query precedente.
È possibile, inoltre, verificare l'appartenenza del valore di un campo ad un elenco di valori. Osserviamo il seguente esempio (Figura 3.7):

```
SELECT OrderId, Freight, CustomerId,  ShipName, OrderDate
FROM Orders
WHERE CustomerId IN ('SUPRD', 'FRANK', 'SPLIR')
```

VEX.Northwind - SQLQuery5.sql* | Table - dbo.Orders | VEX.Northwind - SQLQu

```
SELECT OrderId, Freight, CustomerId,  ShipName, OrderDate
FROM Orders
WHERE CustomerId IN ('SUPRD', 'FRANK', 'SPLIR')
```

Results | Messages

	OrderId	Freight	CustomerId	ShipName	OrderDate
1	10252	51,30	SUPRD	Suprêmes délices	1996-07-09 00:00:00.000
2	10267	208,58	FRANK	Frankenversand	1996-07-29 00:00:00.000
3	10271	4,54	SPLIR	Split Rail Beer & Ale	1996-08-01 00:00:00.000
4	10302	6,27	SUPRD	Suprêmes délices	1996-09-10 00:00:00.000
5	10329	191,67	SPLIR	Split Rail Beer & Ale	1996-10-15 00:00:00.000
6	10337	108,26	FRANK	Frankenversand	1996-10-24 00:00:00.000

Figura 3.8: L'operatore IN... azione

PuntoInformatico Libri

La nuova condizione include nei risultati le sole righe che hanno come valore del campo CustomerId uno dei tre valori indicato nella IN. È l'equivalente di scrivere:

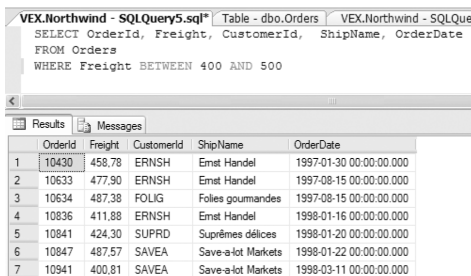
```
SELECT OrderId, Freight, CustomerId,  ShipName, OrderDate
FROM Orders
WHERE CustomerId = 'SUPRD'
          OR CustomerId = 'FRANK'
          OR CustomerId = 'SPLIR'
```

Anche l'operatore IN è abbinabile al NOT che ne inverte il risultato:

```
SELECT OrderId, Freight, CustomerId,  ShipName, OrderDate
FROM Orders
WHERE CustomerId NOT IN ('SUPRD', 'FRANK', 'SPLIR')
```

Se invece i valori accettabili seguono una logica analogica e cioè sono compresi in un range, possiamo usare l'operatore BETWEEN (Figura 3.8):

```
SELECT OrderId, Freight, CustomerId,  ShipName, OrderDate
FROM Orders
WHERE Freight BETWEEN 400 AND 500
```



	OrderId	Freight	CustomerId	ShipName	OrderDate
1	10430	458,78	ERNSH	Ernst Handel	1997-01-30 00:00:00.000
2	10633	477,90	ERNSH	Ernst Handel	1997-08-15 00:00:00.000
3	10634	487,38	FOLIG	Folies gourmandes	1997-08-15 00:00:00.000
4	10836	411,88	ERNSH	Ernst Handel	1998-01-16 00:00:00.000
5	10841	424,30	SUPRD	Suprêmes délices	1998-01-20 00:00:00.000
6	10847	487,57	SAVEA	Save-a-lot Markets	1998-01-22 00:00:00.000
7	10941	400,81	SAVEA	Save-a-lot Markets	1998-03-11 00:00:00.000

Figura 3.9: I confronti con BETWEEN

Con tutta evidenza il BETWEEN può essere sostituito con una ben più prolissa e meno compatta sintassi basta su un doppio AND. Infatti possiamo riscrivere la query precedente nel seguente modo:

```
SELECT OrderId, Freight, CustomerId,  ShipName, OrderDate
FROM Orders
WHERE Freight >= 400 AND Freight <= 500
```

Nel primo capitolo abbiamo introdotto il concetto di NULL e dei comportamenti che esso assume nelle valutazioni. È il caso di spendere qualche parola in più. Ricordiamo che esso esprime l'assenza di un valore e non un valore azzerato (ad esempio uno 0 in un campo numerico o una stringa vuota in un campo testuale). Inoltre si è detto che la presenza anche di un solo NULL in un'espressione rende NULL l'intera espressione. Dunque questo valore va gestito in modo accurato per evitare spiacevoli comportanti inaspettati dalle nostre query. Per testare il valore NULL in una condizione di filtro esiste la parola chiave IS NULL e l'omologo IS NOT NUL. Per la nostra solita query di esempio, questa volta facciamo riferimento ad un'altra tabella di Northwind, Customers (clienti):

```
SELECT CustomerId, CompanyName, Region
FROM Customers
WHERE Region IS NULL
```

Un altro potente operatore di confronto è LIKE che agisce sulle stringhe e permette di effettuare confronti molto più sofisticati di quelli possibili con i normali operatori. La sua sintassi è:

```
colonna LIKE 'pattern'
```

Il pattern può essere semplicemente una stringa, in tal caso funziona esattamente come l'operatore uguale:

PuntoInformatico | Libri

```
SELECT OrderId, Freight, CustomerId,  ShipName, OrderDate
FROM Orders
WHERE CustomerId LIKE 'SUPRD'
```

Oppure può essere un vero e proprio pattern e cioè un modello di stringa contenente anche caratteri jolly speciali, dette wildcard. Essi sono due:

- % (segno di percentuale);
- _ (sottolineatura).

Il primo sostituisce un numero variabile di caratteri, il secondo soltanto un carattere. La seguente query ci offre un esempio del loro uso:

```
SELECT OrderId, CustomerId, EmployeeId, ShipName
FROM Orders
WHERE CustomerId LIKE '_A%A'
```

Il campo CustomerId contiene il codice descrittore del cliente. Con la nostra query intendiamo recuperare i soli clienti che iniziano con una qualsiasi lettera ('_A%A') hanno come seconda lettera 'A' ('_A%A'), poi un porzione di altri caratteri di dimensione variabile ('_A%A') che si chiudono con un'altra 'A' ('_A%A'). Un'altra possibilità di scansione è l'uso dei caratteri [] e ^. Con essi è possibile verificare che un carattere della stringa ad una certa posizione sia compreso in un certo intervallo di caratteri. Vediamo un esempio:

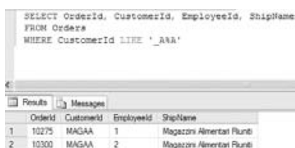


Figura 3.11: L'operatore LIKE e le wildcard

```
SELECT OrderId, CustomerId, EmployeeId, ShipName
FROM Orders
WHERE CustomerId LIKE '[D-K]%A'
```

In pratica si ricercano tutte le stringhe che iniziano con un carattere compreso nell'intervallo D-K, proseguono con una qualsiasi sequenza di caratteri e terminano con A. È possibile anche indicare l'esclusione da un certo intervallo e cioè che un certo carattere non debba rientrare nell'intervallo di caratteri indicato:

```
SELECT OrderId, CustomerId, EmployeeId, ShipName
FROM Orders
WHERE CustomerId LIKE '^[D-K]%A'
```

not connected - SQLQuery5.sql* Table - dbo.Orders VEX.Northwir

```
SELECT OrderId, CustomerId, EmployeeId, ShipName
FROM Orders
WHERE CustomerId LIKE '[D-K]%A'
```

Results Messages

	OrderId	CustomerId	EmployeeId	ShipName
1	10257	HILAA	4	HILARION-Abastos
2	10347	FAMIA	4	Familia Arquibaldo
3	10386	FAMIA	9	Familia Arquibaldo
4	10395	HILAA	6	HILARION-Abastos
5	10414	FAMIA	2	Familia Arquibaldo
6	10476	HILAA	8	HILARION-Abastos
7	10486	HILAA	1	HILARION-Abastos
8	10490	HILAA	7	HILARION-Abastos

Figura 3.12: L'operatore LIKE e i blocchi di caratteri

L'operatore LIKE è abbinabile con l'operatore NOT che, da copione, ne inverte il significato (colonna NOT LIKE 'pattern').

3.2.3 La clausola GROUP BY

Questa clausola opzionale è molto utile perché consente l'aggregazione dei dati risultanti da una query. A volte capita, infatti, soprattutto

PuntoInformatico Libri

per query in JOIN che vedremo in seguito, che nel risultato compaiano delle righe perfettamente identiche, cioè con tutti i campi uguali. Queste ripetizioni spesso sono inutili o dannose perché magari sono l'effetto di una moltiplicazione geometrica di righe rinvenente da una JOIN. Ma questo fenomeno può manifestarsi anche su query semplici basate su una sola tabella, se tra i campi di risultato vengono omessi in tutto o in parte i campi della chiave primaria che sono gli unici che garantiscono l'univocità di valori in tutta la tabella.
Osserviamo un esempio molto semplice basato sulla nostra solita tabella Orders di Northwind. Immagiamo di voler conoscere l'insieme di tutti i clienti (CustomerId) presenti nella tabella Orders, indipendentemente da quanti ordini sono associati a ciascun cliente:

```
SELECT CustomerId
FROM Orders
GROUP BY CustomerId
```

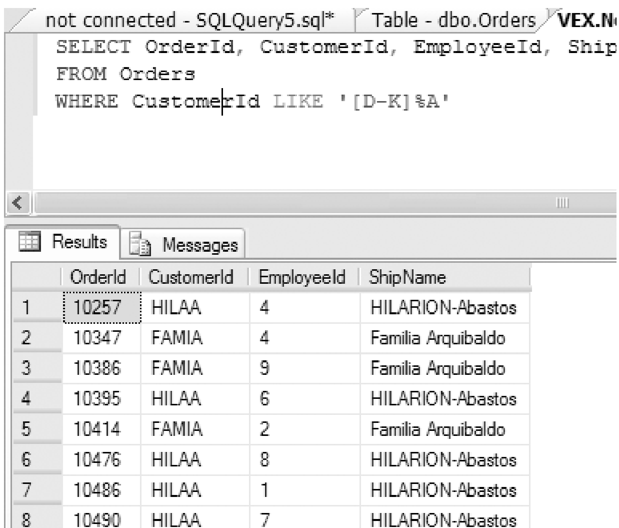


Figura 3.13: La GROUP BY in azione

In Figura 3.13 ne possiamo osservare il risultato: viene riportato semplicemente l'elenco dei clienti presi solo una volta e senza ripetizioni. L'aggregazione ha per sua stessa natura bisogno di un elenco ordinato di dati, pertanto SQL Server, in presenza di una GROUP BY, produce un ordinamento ascendente dei campi raggruppati.
A questo punto sarebbe interessante conoscere il numero di ordini associati a ciascun cliente e dunque il numero di righe di Orders per ciascuno di essi. Per fortuna la GROUP BY consente l'uso di funzioni statistiche, cioè di effettuare statistiche sulle righe raggruppate. La funzione COUNT ci consente proprio di effettuare la nostra conta:

```
SELECT CustomerId, COUNT(*) AS NumeroOrdini
FROM Orders
GROUP BY CustomerId
```

Il risultato sarà proprio il numero di ordini associato al cliente. Tale valore verrà inserito nell'alias NumeroRighe, che potrà essere trattato e fruite come un normalissimo campo calcolato. La funzione COUNT usa nell'esempio come argomento *: esso indica di riportare esattamente il numero di righe raggruppate in base al criterio di GROUP BY senza fare altre considerazioni. Se però volessimo effettuare il calcolo delle occorrenze di un altro campo nelle righe raggruppate, ad esempio se volessimo sapere quanti diversi valori di ShipVia sono presenti nell'ambito nel raggruppamento. Però perché questa statistica sia efficace, dobbiamo dire a SQL Server di considerare i valori distinti di ShipVia e quindi senza ripezioni. A questo serve l'opzione DISTINCT affiancabile all'argomento della COUNT:

```
SELECT CustomerId, COUNT(*), COUNT(DISTINCT ShipVia)
FROM Orders
GROUP BY CustomerId
```

L'opzione di default della COUNT e di tutte le altre funzioni statisti-

PuntoInformatico Libri

che associabili alla GROUP BY e che vedremo di seguito, è ALL (es. SELECT CustomerId, COUNT(ALL ShipVia) FROM Orders GROUP BY CustomerId). Il raggruppamento è possibile anche su due o più campi, come si può osservare nel seguente esempio:

```
SELECT CustomerId, ShipVia, COUNT(*) as NumeroOccorrenze
FROM Orders
GROUP BY CustomerId, ShipVia
```

È importante osservare che nella clausola SELECT potranno essere presenti solo i campi proposti nella clausola GROUP BY, a parte naturalmente i campi calcolati che fanno uso di funzioni statistiche. In caso di raggruppamento su più campi, SQL Server individua tutte le occorrenze con valori distinti dei campi indicati nella tupla definita nella GROUP BY e produce un ordinamento ascendente dei risultati applicati sulle colonne nell'ordine in cui vengono definite nella clausola di raggruppamento. Nell'esempio, dunque, il resultset verrà ordinato prima per CustomerId e poi per ShipVia. È interessante osservare che l'opzione DISTINCT, se usata senza la GROUP BY, si comporta esattamente come quest'ultima con il limite, però, di non poter usare funzioni stastiche. Pertanto la query:

```
SELECT CustomerId
FROM Orders
GROUP BY CustomerId
si può scrive anche come:
SELECT DISTINCT CustomerId
FROM Orders
```

Come si accennava in precedenza, COUNT non è l'unica funzione statistiche fornita da SQL Server. Nella tabella di seguito si riporta l'elenco delle funzioni statistiche. Tutte le funzioni

FUNZIONE	DESCRIZIONE	ESEMPIO
COUNT	Ritorna il numero di occorrenze delle righe incluse nel raggruppamento	SELECT CustomerId, ShipVia, COUNT(*) FROM Orders GROUP BY CustomerId, ShipVia
COUNT_BIG	È identica alla COUNT, ma restituisce valori BIGINT anziché INT	SELECT CustomerId, ShipVia, COUNT_BIG(*) FROM Orders GROUP BY CustomerId, ShipVia
SUM	Questa funzione restituisce la somma di tutti i valori del campo o dell'espressione passato come argomento. I valori sono naturalmente presi tra le righe del raggruppamento corrente. La funziona accetta solo campi numerici e va in errore se vengono passati campi stringa.	SELECT CustomerId, ShipVia, SUM(Freight) FROM Orders GROUP BY CustomerId, ShipVia
MIN	Nell'ambito delle righe raggruppate, la MIN restituisce il minimo valore del campo o dell'espressione passato come argomento. Il suo uso e l'uso di tutte le funzioni che verranno elencate di seguito, ha senso solo se il campo passato come argomento della funzione non è tra i campi di raggruppamento. Questa possibilità, infatti, è contemplata sintatticamente ma non avrebbe senso perché verrebbe restituito sempre lo stesso valore che è invariante per tutti i campi di raggruppamento. La funzione agisce sia su campi numerici (in tal caso va interpretato come il più basso valore dell'insieme) che stringa (in tal caso va interpreto come la prima stringa dell'elenco ordinato in modo ascendente).	SELECT CustomerId, ShipVia, MIN(OrderId) FROM Orders GROUP BY CustomerId, ShipVia
MAX	Nell'ambito delle righe raggruppate, la MAX restituisce il massimo valore del campo o dell'espressione passato come argomento. Valgono le stesse considerazioni fatte per la MIN.	SELECT CustomerId, ShipVia, MAX(OrderId) FROM Orders GROUP BY CustomerId, ShipVia
AVG	Questa funzione restituisce la media di tutti i valori del campo o dell'espressione passata come argomento.	SELECT CustomerId, ShipVia, AVG(Freight), AVG(Freight + 10) FROM Orders GROUP BY CustomerId, ShipVia

© PuntoInformatico Libri

FUNZIONE	DESCRIZIONE	ESEMPIO
STDEV, STDEVP	Queste due funzioni restituiscono la deviazione standard e la deviazione standard di tutta la popolazione di tutti i valori del campo o dell'espressione passata come argomento.	SELECT CustomerId, ShipVia, STDEV(Freight), STDEVP(Freight) FROM Orders GROUP BY CustomerId, ShipVia
VAR, VARP	Queste due funzioni restituiscono la varianza statistica e la varianza statistica di tutta la popolazione di tutti i valori del campo o dell'espressione passata come argomento.	SELECT CustomerId, ShipVia, VAR(Freight), VARP(Freight) FROM Orders GROUP BY CustomerId, ShipVia

3.2.4 La clausola HAVING

Questa clausola opzionale si abbina alla clausola GROUP BY, in assenza della quale perderebbe di significato nonostante ne sia comunque possibile l'uso (in tal caso agisce come se fosse una WHERE aggiuntiva perché considera il le righe restituite dalla query come appartenenti ad un solo gruppo). Si può, in linea di massima, considerare una seconda clasola WHERE che però agisce solo sugli effetti del raggruppamento operato dalla GROUP BY agendo quindi sui campi interessati al raggruppamento o sui risultati delle funzioni statistiche invocabili grazie alla presenza del raggruppamento. Osserviamo la seguente query:

```
SELECT Shipvia, CustomerId, COUNT(*) AS NumeroRighe, MAX(OrderId) AS
                                                              OrderIdMax
FROM Orders
GROUP BY Shipvia, CustomerId
HAVING Shipvia = 1
```

Questo esempio mostra un utilizzo banale e probabilmente inappropriato della HAVING perché la condizione che è stata inserita poteva benissimo essere inserita nella WHERE, senza scomodare la HAVING e, indirettamente, la GROUP BY. Tuttavia sintatticamente è una modalità

corretta. Invece, l'uso più appropriato che si può fare di questa clausola è abbinato alle funzioni statistiche. Eccone un esempio:

```
SELECT Shipvia, CustomerId, COUNT(*) AS NumeroRighe, MAX(OrderId) AS
                                                          OrderIdMax
FROM Orders
GROUP BY Shipvia, CustomerId
HAVING COUNT(*) > 8
```

Questo esempio mostra la potenza della clasola che consente, ad esempio, in una query con raggruppamento, di filtrare le sole righe che hanno almeno 8 occorenze a seguito del raggruppamento. In generale è possibile adoperare tutte le funzioni statistiche abbinabili alla GROUP BY e dunque COUNT, MAX, MIN, AVG, ecc...

3.2.5 La clausola ORDER BY

Questa clausola consente di effettuare l'ordinamento del resultset restituito da una query. È opzionale. Infatti, se non indicata, il resultset della query viene restituito secondo l'ordinamento naturale, cioè secondo la posizione fisica dei record nelle tabelle di origine della query, a meno che non ci sia una clasola GROUP BY o un'opzione DISTINCT sulla clausola SELECT che portano automaticamente ad un ordinamento del risultato basato sui campi della DISTINCT o della GROUP BY. L'ordinamento si basa sui valori dei campi della SELECT stessa anche se i campi che contribuiscono all'ordinamento non devono essere necesssiamente presenti nella clausola SELECT, a meno che non ci sia un'opzione DISTINCT o non sia presente una clausola GROUP BY. In tal caso, i campi usati nell'ordinamento devono essere anche restitito dalla query. Osserviamone la sintassi:

```
ORDER BY colonna_1 [DESC] [, colonna_2 [DESC] [... , colonna_n [DESC]]]
```

In presenza di ordinamento su più campi, SQL Server ordina prima

PuntoInformatico Libri

il risultato per il primo campo a partire da sinistra, poi raffina l'ordinamento sul secondo campo, qualora dopo il primo ordinamento ci siano righe con posizione equivalente, e così via per tutti gli altri campi indicati nella clausola. Il criterio naturale di ordinamento è quello crescente, però è possibile indicare, accanto a ciascun campo, l'opzione DESC che ne inverte l'ordine trasformandolo in ordinamento decrescente. Si badi bene che tale opzione si applica solo sul campo su cui viene indicata e non sull'interno ordinamento, pertanto sarà possibile avere criteri misti di ordinamento, crescente e decrescente, sulla stessa query.
Osserviamo il seguente esempio per una migliore comprensione:

```
SELECT OrderId, CustomerId, EmployeeId, ShipName
FROM Orders
WHERE CustomerId LIKE '[D-K]%A'
ORDER BY CustomerId, EmployeeId DESC
```

Nell'esempio l'ordinamento del risultato agisce primariamente sul campo CustomerId e poi, in modo discendente, sul campo campo EmployeeId.

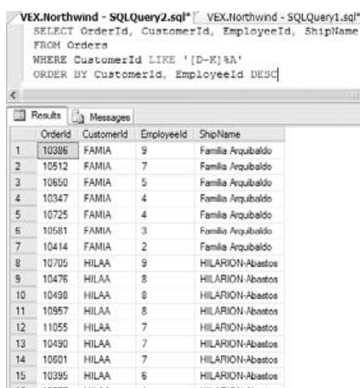


Figura 3.14: La clausola ORDER BY

È poi possibile effettuare l'ordinamento del risultato riferendosi direttamente ai campi della clausola SELECT in modo posizionale. Si osservi l'esempio:

```
SELECT OrderId, CustomerId, EmployeeId, ShipName
FROM Orders
ORDER BY 3, 2 DESC
```

In pratica si invoca l'ordinamento sul terzo campo della SELECT (EmployeeId) e poi sul secondo in modalità discendente (CustomerId). Questa possibilità, se da un lato introduce ambiguità e rende meno leggibili le query oltre che potenzialmente deboli, perché basta cambiare involontariamente l'ordine delle colonne nella clusola SELECT per cambiare il senso di ordinamento della query, dall'altro offre un impareggiabile vantaggio quando nelle SELECT ci sono campi calcolati. In tal caso, infatti, sarebbe necessario rieffettuare il calcolo del campo anche nella ORDER BY per poterne effetture l'ordinamento. Osserviamo questo esempio:

```
SELECT OrderId, Freight, Freight * 0.727 as EuroFreight
FROM Orders
ORDER BY Freight * 0.727
```

Questa versione è piuttosto onerosa perché il calcolo del campo deve essere effettuato due volte: sia nella clausola SELECT che nella clausola ORDER BY. Invece è possibile semplificarla in questo modo:

```
SELECT OrderId, Freight, Freight * 0.727 as EuroFreight
FROM Orders
ORDER BY 3
```

3.2.6 La clausola COMPUTE

La clausola COMPUTE e la sua variante COMPUTE BY non appar-

PuntoInformatico Libri

tengono al linguaggio SQL standard, ma sono un'estensione specifica di SQL Server in grado di generare i totali per alcune righe della query. Questi valori non vengono inclusi nel resultset della query ma sono esterni ad esso, pertanto il suo scopo è limitato nella reportistica.
Osserviamo la seguente query:

```
SELECT CustomerId, ShipVia, Freight
FROM Orders
COMPUTE SUM(Freight)
```

Figura 3.15: La clausola ORDER BY

Questa query viene eseguita regularmente secondo le sue clausole SELECT e FROM In Figura3.15 possiamo osservarne il risultato: vediamo, al termine del resultset, una sezione separata con un solo campo SUM che contiene la somma di tutti i valori Freight individuati dalla query. È inoltre possibile effettuare anche subtotali, cioè somme parziali: in tal caso si fa uso della clausola COMPUTE BY che va necessariamente coordinata con ORDER BY. Vediamone un esempio:

```
SELECT CustomerId, ShipVia, Freight
FROM Orders
ORDER BY CustomerId
COMPUTE SUM(Freight) BY CustomerId
```

In questo caso non viene stampato il totale generale di Freight, ma una serie di subtotali dello stesso campo, ottenuti per rottura di codice sulla tupla indicata dalla ORDER BY.

3.3 RAGIONARE PER INSIEMI

In questo lungo capitolo abbiamo finora esaminato ogni possibile operazione di interrogazione effettuabile su una singola tabella. Ma questo approccio è troppo limitante perché, per effetto delle regole di normalizzazione dei database, qualsiasi informazione, anche se molto semplice, spesso è distribuita su più tabelle. Pensiamo ad un ordine costituito da una tabella di testata e una tabella di dettaglio. La tabella di testata riporta il riferimento al codice cliente, ma le informazioni del cliente sono contenute nella tabella di anagrafica clienti e pertanto sulla testata d'ordine di sarà una foreign key verso i clienti. Stesso discorso vale per le righe: ciascuna di essere recherà il codice dell'articolo in ordine, ma le informazioni complete che lo contraddistingono (descrizione, unità di misura, gruppo merceologico, produttore ecc...) sono dislocate nella tabelle delle anagrafiche articoli sui cui la tabella delle righe è in foreign key. Pertanto, se volessimo conoscere l'elenco l'elenco di tutti i tipi di articoli messi in ordini in un certo mese e per ciascuno di essi conoscerne il codice e la descrizione, la quantità venduta e il codice e la descrizione del cliente destinatario dell'ordine, dovremmo accedere a quattro tabelle recuperando le varie informazioni da restituire nella nostra query. In SQL tutto questo è possibile ed è proprio questa caratteristica che fa la differenza rispetto, ad esempio, all'accesso ai file binari o testo alla PASCAL o alla C o ad altre forme più rudimentali di memorizzazione dei dati. Con questi altri sistemi saremmo costretti ad aprire cia-

PuntoInformatico | Libri

scuno dei file contenti le quattro informazioni, individuare e leggere da ciascuna di esse i (pochi) campi a noi necessari e assemblare nel nostro codice una struttura contente le informazioni da restituire popolate in modalità manuale.

3.3.1 La teoria

Ormai sappiamo che il modello relazionale inventato da Codd si basa anche sulla teoria degli insiemi. Le tre più comuni operazioni degli insiemi sono:

- *Intersezione* – Si definisce intersezione fra A e B il sottoinsieme formato dagli elementi comuni agli insiemi A e B; vengono usate quando si vogliono individuare elementi comuni a due insiemi (ad esempio, tra gli alunni di due classi, che quindi rappresentano due diversi insiemi, individuare nei due gruppi gli alunni di sesso femminali e che sono nati in febbraio);
- *Differenza* – Si definisce differenza fra A e B, dati in questo ordine, il sottoinsieme formato dagli elementi di A che non appartengono a B; vengono usate per individuare elementi non comuni ad entrambi i gruppi (ad esempio, tra gli alunni di due classi, individuare l'elenco di tutti i maschi della prima classe che hanno un nome di battesimo che non è anche presente in nessuno dei maschi della seconda classe);
- *Unione* – Si definisce unione fra A e B l'insieme formato dagli elementi che appartengono almeno a uno dei due insiemi A e B; vengono usate quando si vogliono individuare l'insieme di tutti gli elementi presenti nei due insiemi due insiemi (ad esempio, tra gli animali mammiferi e gli animali che vivono in acqua, che quindi rappresentano due diversi insiemi, individuare l'insieme complessivo degli animali; questo insieme può presentare anche delle ripetizioni cioè elementi presenti in entrambi gli insiemi – si pensi al delfino – ma l'operazione di unione li include entrambi).

3.4 GLI OPERATORI DI INSIEME

SQL Server mette a disposizione degli operatori insiemistici, da applicare nella scrittura delle nostre interrogazioni. Tali operatori operano sul risultato di più select. Gli attributi interessati dagli operatori di insieme devono esser di tipo compatibile tra loro. Gli operatori disponibili sono gli operatori di UNION (unione), INTERSECT (intersezione) e EXCEPT (differenza), il significato è analogo ai corrispondenti operatori dell'algebra insiemistica.

3.4.1 Operatori UNION e UNION ALL

L'operatore UNION restituisce il risultato di più query escludendo le righe duplicate, vediamo un esempio: facendo riferimento al database di esempio Northwind, consideriamo le due tabelle Customers (l'elenco dei clienti) e Suppliers (l'elenco dei fornitori), come mostrato in Figura 3.16.

Customers

Column Name	Data Type	Allow Nulls
CustomerID	nchar(5)	☐
CompanyName	nvarchar(40)	☐
ContactName	nvarchar(30)	☑
ContactTitle	nvarchar(30)	☑
Address	nvarchar(60)	☑
City	nvarchar(15)	☑
Region	nvarchar(15)	☑
PostalCode	nvarchar(10)	☑
Country	nvarchar(15)	☑
Phone	nvarchar(24)	☑
Fax	nvarchar(24)	☑
		☐

Suppliers

Column Name	Data Type	Allow Nulls
SupplierID	int	☐
CompanyName	nvarchar(40)	☐
ContactName	nvarchar(30)	☑
ContactTitle	nvarchar(30)	☑
Address	nvarchar(60)	☑
City	nvarchar(15)	☑
Region	nvarchar(15)	☑
PostalCode	nvarchar(10)	☑
Country	nvarchar(15)	☑
Phone	nvarchar(24)	☑
Fax	nvarchar(24)	☑
HomePage	ntext	☑
		☐

Figura 3.16: Struttura delle tabelle Customers e Suppliers

Ciascuna delle due tabelle anagrafice, tra le altre informazioni riportano anche i campi City e Country, rispettivamrnte la città e la nazione dei soggetti. Vogliamo innanzitutto estrarre l'elenco di tutte le città e dunque di tutte le coppie, senza ripetione, di campi City e

PuntoInformatico Libri

Country da ciascuna delle due tabelle. Lo facciamo con normali SELECT DISTINCT
Vogliamo l'elenco quali persone giocano in una squadra o nell'altra:

```
SELECT DISTINCT City, Country
FROM Suppliers
GO
SELECT DISTINCT City, Country
FROM Customers
GO
```

Nelle query di esempio ci sono 29 città nei fornitori e 69 nei clienti. Ma se volessimo ottenere in una sola query l'elenco di tutte le coppie City Country che compaiono almeno una volta tra i fornitori o tra i clienti, dovremmo ricorrere al costrutto UNION:

```
SELECT DISTINCT City, Country
FROM Suppliers
UNION
SELECT DISTINCT City, Country
FROM Customers
```

La sintassi della UNION prevede che si possano eseguire due o più query collegate tra loro dal costrutto UNION; l'unico requisito è che ogni query restituisca sempre lo stesso numero di campi, dello stesso tipo, nella stessa posizione. Questo non significa che, restando all'esempio, tutte le query debbano restituire necessariamente un campo City di tipo stringa e un campo Country di tipo stringa, ma solo che tutte debbano restituire due campi stringa, eventualmente anche con nomi diversi. In realtà, tra le 29 città dei fornitori e le 69 dei clienti, ve ne sono solo 5 in comune. Eccole:

```
City          Country
```

```
------------------------
Berlin      Germany
London         UK
Montréal       Canada
Paris       France
Sao Paulo      Brazil
```

Dunque, la UNION, considera questi elementi una sola volta portando il totale di righe a 93 anziché a 98. Ma se volessimo considerare anche le ripetizioni, avremmo a disposizione il comando UNION ALL:

```
SELECT DISTINCT City, Country
FROM Suppliers
UNION ALL
SELECT DISTINCT City, Country
FROM Customers
```

La UNION è il più importante operatore di insiemi ed è l'unico dei tre che non può essere simulato con gli altri operatori di aggregazioni basati sulla JOIN che vedremo in seguito ed è l'unico operatore implementato da SQL Server fino alla version 2000.

l Il nuovo operatore INTERSECT

SQL Server 2005, invece, introduce l'operatore INTERSECT che ci restituisce proprio i 5 record comuni alle due query. Vediamone la sintassi applicata al nostro esempio:

```
SELECT DISTINCT City, Country
FROM Suppliers
INTERSECT
SELECT DISTINCT City, Country
FROM Customers
```

Questo operatore può essere facilmente implementato da una bana-

PuntoInformatico Libri

le INNER JOIN, come vedremo nel proseguo.

3.4.2 Il nuovo operatore EXCEPT

L'operatore EXCEPT, infine, restituisce i soli record presenti nel primo insieme ma assenti nel secondo. Osserviamone la sintassi:

```
SELECT DISTINCT City, Country
FROM Suppliers
EXCEPT
SELECT DISTINCT City, Country
FROM Customers
```

Quale risultato ci restituirà la query di esempio? Sappiamo che Suppliers contiene 69 record e che 5 di questi sono in comune con Customers. Dunque il risultato sarà esattamente costutito dalle 64 (69 – 5) righe presenti in Suppliers e assenti in Customers. A differenza degli operatori UNION, UNION ALL e INTERSECT, che sono non posizionali, l'operatore EXCEPT lo è nel senso che invertendo l'ordine delle query parziali il risultato cambia. Infatti, se riscrivessimo la query al contrario:

```
SELECT DISTINCT City, Country
FROM Suppliers
EXCEPT
SELECT DISTINCT City, Country
FROM Customers
```

Le righe restituite sarebbe diverse in valore e in numero e sarebbero esattamente 24 (29 – 5) perché l'insieme di riferimento diventa Customers e non più Suppliers.

3.5 COMBINAZIONE DI TABELLE

L'operazione più importante che interessa combinazioni di tabelle

diverse è la JOIN. Essa sfrutta pienamente le caratteristiche relazionali del database. Significa letteralmente unione di tabelle. Esistono vari tipi di join, ma tutti derivano o possono essere ricondotti a vari operatori dell'algebra insiemistica. L'importanza principale del join risiede nella possibilità che ci offre per correlare e visualizzare dati appartenenti a tabelle diverse o alla medesima tabella, logicamente correlati tra di loro. I semplici dati così uniti possono assumere la forma di complesse informazioni così come noi li vogliamo.

3.5.1 CROSS JOIN

Per comprendere a pieno l'operazione di CROSS JOIN, cioè di unione incrociata bisogna aver ben chiaro il concetto di prodotto cartesiano: dati due insiemi A e B si chiama prodotto cartesiano di A e B, l'insieme delle coppie ordinate (v1, v2), tali che v1 è un elemento di A e v2 un elemento di B. Consideriamo i valori contenuti nei due insiemi:

```
A = {'Bari', 'Londra', 'New York'}
B = {'chiesa', 'scuola', 'municipio', 'teatro'}
```

Il prodotto cartesiamo dei due insiemi sarà:

```
A x B = {('Bari', 'chiesa'), ('Bari', 'scuola'), ('Bari', 'municipio'), ('Bari', 'tea
        tro'), ('Londra', 'chiesa'), ('Londra', 'scuola'), ('Londra', 'municipio'),
     ('Londra', 'teatro'), ('New York', 'chiesa'), ('New York', 'scuola'), ('New
                              York', 'municipio'), ('New York', 'teatro')}
```

Il prodotto cartesiano fra i due insiemi è dato da tutti gli elementi di A combinati con ogni elemento di B. Nell'esempio abbiamo usato solo due insiemi ma il prodotto cartesiano è applicabile anche a più di due insiemi.
Considerando che le tabelle non sono altro che insiemi i cui elementi sono le righe ecco che possiamo individuare l'operazione di CROSS

PuntoInformatico Libri

JOIN in quella di prodotto cartesiano appartenente alle teorie degli insiemi. Dunque il prodotto cartesiano tra due o più tabelle si traduce in una istruzione chiamata CROSS JOIN. Il CROSS JOIN si ottiene in maniera molto semplice elencando dopo la FROM le tabelle che devono essere coinvolte. Vediamo la sintassi SQL applicata all'esempio con le due tabelle Citta e Edifici:

```
SELECT Citta.NomeCitta, Edifici.TipoEdificio
FROM Citta, Edifici
```

Il risultato sarà esattamente quello descritto nello sviluppo dell'esempio precedente. La sintassi appena indicata è però da considerarsi obsoleta ed è, invece, da preferire la sintassi esplicita:

```
SELECT Citta.NomeCitta, Edifici.TipoEdificio
FROM Citta
CROS JOIN Edifici
```

Il CROSS JOIN non è particolarmente utile e viene usato raramente, ma abbinato alla clausola WHERE può acquistare un maggior senso pratico.

3.5.2 INNER JOIN

Il INNER JOIN è un tipo di operazione che ci permette di correlare due o più tabelle sulla base di valori uguali in attributi contenenti lo stesso tipo di dati. Riprendiamo il precedente esempio di City, Country basato sulle tabelle Suppliers e Customers di Northwind e usato per scrivere gli operatori insiemistici. Tali esempi, infatti, si prestavo perfettamente. Vediamo un esempio:

```
SELECT Suppliers.SupplierId, Suppliers.City AS SupplierCity,
                                        Suppliers.Country AS SupplierCountry,
Customers.CustomerId, Customers.City AS CustomerCity,
```

```
                    Customers.Country AS CustomerCountry
FROM Suppliers, Customers
WHERE Suppliers.City = Customers.City
    AND Suppliers.Country = Customers.Country
```

Abbiamo incrociato le tabelle Customers e Suppliers attraverso la condizione Suppliers.City = Customers.City e Suppliers.Country = Customers.Country e ne abbiamo rispettivamente preso i campi SupplierId, City, Country e CustomerId, City, Country come mostrato in Figura 3.17.

```
SELECT Suppliers.SupplierId, Suppliers.City, Suppliers.Country,
Customers.CustomerId, Customers.City, Customers.Country
FROM Suppliers, Customers
WHERE Suppliers.City = Customers.City
      AND Suppliers.Country = Customers.Country
```

Results | Messages

	SupplierId	City	Country	CustomerId	City	Country
1	1	London	UK	AROUT	London	UK
2	1	London	UK	BSBEV	London	UK
3	1	London	UK	CONSH	London	UK
4	1	London	UK	EASTC	London	UK
5	1	London	UK	NORTS	London	UK
6	1	London	UK	SEVES	London	UK
7	10	Sao Paulo	Brazil	COMMI	Sao Paulo	Brazil
8	10	Sao Paulo	Brazil	FAMIA	Sao Paulo	Brazil
9	10	Sao Paulo	Brazil	QUEEN	Sao Paulo	Brazil
10	10	Sao Paulo	Brazil	TRADH	Sao Paulo	Brazil
11	11	Berlin	Germany	ALFKI	Berlin	Germany
12	18	Paris	France	PARIS	Paris	France
13	18	Paris	France	SPECD	Paris	France
14	25	Montréal	Canada	MEREP	Montréal	Canada

Figura 3.17: La INNER JOIN azione

La novità di questa nuova query è che la clausola FROM contiene due tabelle e non più una, come abbiamo visto in tutte gli esempi precedenti. È infatti possibile eseguire interrogazioni anche su due o più tabelle separandole virgole. Le due tabelle dell'esempio hanno in comune i campi Country e City e perciò nella clasuola WHERE vengono correlate con la condizione WHERE Suppliers.City = Customers.City AND Suppliers.Country = Customers.Country. È altresì interessante osservare come, per evitare ambiguità, ciascun campo è indicato con la sintassi completa di table space e cioè NomeTabella.NomeCam-

PuntoInformatico Libri

po. In generale questa sintassi si adopera quando un campo interessato dalla query, indipendentemente dalla clausola in cui compare (SELECT, WHERE, GROUP BY o ORDER BY), è presente su due o più tabelle interessate dall'interrogazione. L'indicatore di tabella indica, dunque, esattamente da quale tabella recuperare il campo. Se nella query si vuole recuperare nella clausola SELECT più campi omonomi, si rende necessario adoperare gli alias.
La sintassi appena mostrata è però da ritenere obsoleta perché non facente parte dello standard ANSI SQL e perché specifica di SQL Server. La sintassi standard prevede, invece, l'introduzione della parola chiave JOIN. Vediamo la la precedente query di esempio riscritta nella nuova sintassi:

```
SELECT Suppliers.SupplierId, Suppliers.City AS SupplierCity,
                                        Suppliers.Country AS SupplierCountry,
Customers.CustomerId, Customers.City AS CustomerCity,
                                        Customers.Country AS CustomerCountry
FROM Suppliers
INNER JOIN Customers ON
      Suppliers.City = Customers.City
      AND Suppliers.Country = Customers.Country
```

La clausola INNER JOIN va indicata dopo la clausola FROM che torna ad avere una sola tabella. La sintassi è dunque INNER JOIN <nome_tabella_di_join> ON <condizione>. La condizione segue le stesse regole valide per la clausola WHERE. Nel caso della INNER JOIN diventa irrilevante quale delle due tabelle (nell'esempio Customers e Suppliers) compaia nella clausola FROM e quale nella clausola JOIN: il risultato sarà identico e dunque potremmo riscrivere l'esempio come:

```
SELECT Suppliers.SupplierId, Suppliers.City AS SupplierCity,
                                        Suppliers.Country AS SupplierCountry,
```

```
Customers.CustomerId, Customers.City AS CustomerCity,
                                Customers.Country AS CustomerCountry
FROM Customers
INNER JOIN Suppliers ON
    Suppliers.City = Customers.City
    AND Suppliers.Country = Customers.Country
```

In questa nuova sintassi la clausola WHERE è sempre possibile. Osserviamo il seguente esempio:

```
SELECT Suppliers.SupplierId, Suppliers.City AS SupplierCity,
                                Suppliers.Country AS SupplierCountry,
Customers.CustomerId, Customers.City AS CustomerCity,
                                Customers.Country AS CustomerCountry
FROM Customers
INNER JOIN Suppliers ON
    Suppliers.City = Customers.City
    AND Suppliers.Country = Customers.Country
WHERE Customers.CustomerId LIKE 'A%'
```

Ma il campo CustomerId fa in qualche modo parte della clausola JOIN visto che ha a che fare con la correlazione delle due tabelle, pertanto potremmo riscrivere il precedente esempio nel seguente modo e senza che ci siano differenze nel resultset:

```
SELECT Suppliers.SupplierId, Suppliers.City AS SupplierCity,
                                Suppliers.Country AS SupplierCountry,
Customers.CustomerId, Customers.City AS CustomerCity,
                                Customers.Country AS CustomerCountry
FROM Customers
INNER JOIN Suppliers ON
    Suppliers.City = Customers.City
    AND Suppliers.Country = Customers.Country
```

PuntoInformatico Libri

```
    AND Customers.CustomerId LIKE 'A%'
```

Allora quando usare l'una e quando l'altra? Nella INNER JOIN non fa differenza in termini di risultato (nel proseguo vedremo come negli altri tipi di JOIN, invece, cambi tutto). Si tende preferire la seconda versione perché è più leggibile e perché agevola il motore di query di SQL Server nelle sue ottimizzazioni, almeno per le interrogazioni più complesse.
Sebbene non molto usata, da indicare la possibilità che le condizioni espresse nella parte ON della clausola JOIN non esprimano uguaglianza ma segni diversi (ad esempio, ON Suppliers.City <> Customers.City).
Osserviamo, infine, un'interessante possibilità: sulla base dell'esempio precedente, vogliamo ottenere tutte le coppie Country – City, senza ritpetizioni, che sono contemporaneamente presenti nelle due tabelle. Ecco la INNER JOIN che fa il lavoro per noi:

```
SELECT DISTINCT Customers.City, Customers.Country
FROM Customers
INNER JOIN Suppliers ON
    Suppliers.City = Customers.City
    AND Suppliers.Country = Customers.Country
```

La clausola SELECT contiene l'opzione DISTINCT per eliminare le duplicazioni. I campi recuperati sono City e Country presi da Customers, ma avremmo potuto prenderli anche da Suppliers visto che sono presi identici nella ON. Il risultato della query è stranamente identico a quello ottenuto nell'esempio di INTERSECT visto in precedenza... Naturalmente non è un caso, ma la clausola insiemistica INTERSECT si può rendere con una INNER JOIN. Che è proprio quello che si è fatto fino alla versione SQL Server 2000 dato che la INTERSECT era assente. In realtà, on SQL Server 2005 è da preferire quest'ultima perché è più veloce.

3.5.3 OUTER JOIN

Con l'OUTER JOIN è possibile estrapolare anche quei dati, appartenenti ad una delle tabelle, che non verrebbero estrapolati nei tipi di join visti fino a questo momento. Infatti OUTER significa esterno: dati esterni al normale tipo di join. Si rende necessario specificare qual è la tabella di cui vogliamo estrapolare i dati anche se non soddisfano la condizione di join, questo lo facciamo indicando con LEFT o RIGHT se la tabella in questione è quella che appare a destra o a sinistra del comando JOIN.

```
SELECT <campi>
FROM tabella1
[LEFT | RIGHT] JOIN tabella2
      ON <condizione>
```

Vediamo un esempio basato sulla query precedente; vogliamo ottenere tutte le coppie Country, City che sono presenti in Customers ma che in Suppliers potrebbero ANCHE essere assenti:

```
SELECT DISTINCT Suppliers.SupplierId, Suppliers.City AS SupplierCity,
                                    Suppliers.Country AS SupplierCountry,
Customers.CustomerId, Customers.City AS CustomerCity,
                                  Customers.Country AS CustomerCountry
FROM  Suppliers
LEFT OUTER JOIN Customers ON
      Suppliers.City = Customers.City
      AND Suppliers.Country = Customers.Country
```

Significa che sicuramente i campi Suppliers.SupplierId, Suppliers.City e Suppliers.Country saranno sempre presenti perché il record Suppliers è in LEFT OUTER JOIN rispetto a Customers. Osserviamo la Figura 3.18: tutti i campi appartenenti a Suppliers non sono mai a NULL, invece i campi di Customers in alcuni casi sono a NULL. Si consideri che

PuntoInformatico | Libri

il campo CustomerId è chiave primaria della tabella Customers pertanto quando assume il valore NULL nel resultset sta ad indicare che, per quella riga, è stata individuata nella tabella Suppliers ma non in Customers, ragion per cui i campi del resultset sono rimasti NULL che, come ricorderemo, indica proprio l'assenza di valore.

VEX.Northwind - SQLQuery1.sql* | Summary

```
SELECT DISTINCT Suppliers.SupplierId, Suppliers.City AS SupplierCity,
Suppliers.Country AS SupplierCountry,
Customers.CustomerId, Customers.City AS CustomerCity,
Customers.Country AS CustomerCountry
FROM  Suppliers
LEFT OUTER JOIN Customers ON
    Suppliers.City = Customers.City
        AND Suppliers.Country = Customers.Country
```

Results | Messages

	SupplierId	SupplierCity	SupplierCountry	CustomerId	CustomerCity	CustomerCountry
1	1	London	UK	AROUT	London	UK
2	1	London	UK	BSBEV	London	UK
3	1	London	UK	CONSH	London	UK
4	1	London	UK	EASTC	London	UK
5	1	London	UK	NORTS	London	UK
6	1	London	UK	SEVES	London	UK
7	2	New Orleans	USA	NULL	NULL	NULL
8	3	Ann Arbor	USA	NULL	NULL	NULL
9	4	Tokyo	Japan	NULL	NULL	NULL
10	5	Oviedo	Spain	NULL	NULL	NULL
11	6	Osaka	Japan	NULL	NULL	NULL
12	7	Melbourne	Australia	NULL	NULL	NULL
13	8	Manchester	UK	NULL	NULL	NULL
14	9	Göteborg	Sweden	NULL	NULL	NULL
15	10	Sao Paulo	Brazil	COMMI	Sao Paulo	Brazil
16	10	Sao Paulo	Brazil	FAMIA	Sao Paulo	Brazil
17	10	Sao Paulo	Brazil	QUEEN	Sao Paulo	Brazil
18	10	Sao Paulo	Brazil	TRADH	Sao Paulo	Brazil
19	11	Berlin	Germany	ALFKI	Berlin	Germany
20	12	Frankfurt	Germany	NULL	NULL	NULL

Figura 3.18: La LEFT OUTER JOIN azione

Se adoperiamo la versione RIGHT, il significato cambia in modo sensibile perché la tabella di riferimento, la cui presenza nel resultset dovrà essere sempre obbligatoria, diventa quella di destra:

```
SELECT DISTINCT Suppliers.SupplierId, Suppliers.City AS SupplierCity,
                                        Suppliers.Country AS SupplierCountry,
Customers.CustomerId, Customers.City AS CustomerCity,
                                        Customers.Country AS CustomerCountry
FROM  Suppliers
RIGHT OUTER JOIN Customers ON
      Suppliers.City = Customers.City
      AND Suppliers.Country = Customers.Country
```

In Figura 3.19 possiamo osservare il risultato. Invertendo l'ordine

delle tabelle, si inverte anche il significato della OUTER JOIN pertanto, se nella versione LEFT invertiamo Customers con Suppliers, essa diventa una RIGHT e viceversa.

VEX.Northwind - SQLQuery1.sql* Summary

```
SELECT DISTINCT Suppliers.SupplierId, Suppliers.City AS SupplierCity,
Suppliers.Country AS SupplierCountry,
Customers.CustomerId, Customers.City AS CustomerCity,
Customers.Country AS CustomerCountry
FROM  Suppliers
RIGHT OUTER JOIN Customers ON
    Suppliers.City = Customers.City
        AND Suppliers.Country = Customers.Country
```

Results | Messages

	SupplierId	SupplierCity	SupplierCountry	CustomerId	CustomerCity	CustomerCountry
1	11	Berlin	Germany	ALFKI	Berlin	Germany
2	NULL	NULL	NULL	ANATR	México D.F.	Mexico
3	NULL	NULL	NULL	ANTON	México D.F.	Mexico
4	1	London	UK	AROUT	London	UK
5	NULL	NULL	NULL	BERGS	Luleå	Sweden
6	NULL	NULL	NULL	BLAUS	Mannheim	Germany
7	NULL	NULL	NULL	BLONP	Strasbourg	France
8	NULL	NULL	NULL	BOLID	Madrid	Spain
9	NULL	NULL	NULL	BONAP	Marseille	France
10	NULL	NULL	NULL	BOTTM	Tsawassen	Canada
11	1	London	UK	BSBEV	London	UK
12	NULL	NULL	NULL	CACTU	Buenos Aires	Argentina
13	NULL	NULL	NULL	CENTC	México D.F.	Mexico
14	NULL	NULL	NULL	CHOPS	Bern	Switzerland
15	10	Sao Paulo	Brasil	COMMI	Sao Paulo	Brasil
16	1	London	UK	CONSH	London	UK
17	NULL	NULL	NULL	DRACD	Aachen	Germany
18	NULL	NULL	NULL	DUMON	Nantes	France
19	1	London	UK	EASTC	London	UK
20	NULL	NULL	NULL	ERNSH	Graz	Austria

Figura 3.19: La RIGHT OUTER JOIN azione

La OUTER JOIN accetta comunque la WHERE ma, a differenza della INNER JOIN, mettere una condizione nella clausola WHERE anziché nella clausola ON è molto diverso. Infatti, anche in presenza di una OUTER JOIN, le righe che non superano la condizione di WHERE vengono tagliate. Invece, se la stessa condizione è posta nella ON, vengono comunque prese le righe della tabella di sinistra (LEFT) o di destra (RIGHT). Si osservi il seguente esempio:

```
SELECT DISTINCT Customers.City, Customers.Country
FROM Customers
LEFT OUTER JOIN Suppliers ON
    Suppliers.City = Customers.City
    AND Suppliers.Country = Customers.Country
WHERE Suppliers.SupplierId IS NULL
```

Va così interpretato: prendere tutte le coppie City – Country da Cu-

stomers che sono presenti solo in Customers e che sono assenti in Suppliers. Infatti la clausola WHERE, che esprime la condizione SuppliersId IS NULL, serve a tagliare proprio quelle righe del resultset che presentano solo campi sia di Customers che di Suppliers imponendo come regola l'assenza di campi nella seconda tabella, condizione esprimibile testando il NULL, come abbiamo osservato in precedenza. Il risultato della query di esempio è costituito da 64 righe che sono esattamente le stesse restituite dalla EXCEPT che, dunque, è implementabile anche in quest'altra forma. Non stupirà sapere che questa era l'unica modalità possibile fino a SQL Server 2000.

3.5.4 Subquery

Essa è una query che sta all'interno di un'altra interrogazione. La query interna passa i risultati alla query esterna che li verifica nella condizione che segue la clausola WHERE. Osserviamo il seguente esempio:

```
SELECT Orders.OrderId, Orders.ShipName,
Details.ProductId, Details.UnitPrice
FROM Orders
INNER JOIN [Order Details] AS Details
      ON Details.OrderId = Orders.OrderId
WHERE Details.UnitPrice =
        (
              SELECT MAX(UnitPrice)
              FROM [Order Details] AS DetSub
              WHERE DetSub.OrderId = Orders.OrderId
        )
```

Si tratta di una classa INNER JOIN tra la tabella Orders e Order Details di Northwind che restituisce i campi OrderId, ShipName (dalla testata), ProductId ed UnitPrice (dalle righe) che, senza la parte WHERE, restituirebbe, per ciascun ordine di Orders, tutte le righe. La con-

dizione di WHERE, però, include una subquery che, di ciascun ordine, recupera la sola riga con il massimo valore di UnitPrice. Avremo così, per ciascun ordine, una ed una sola riga di dettaglio contente il prodotto con il maggior prezzo unitario per quell'ordine.

Figura 3.20: Una subquery che ritorna un solo valore

La subquery che abbiamo appena visto ritorna un solo valore, ma è possibile definire anche subquery che ritornito resultset. Osserviamo un esempio leggermente modificato della query precedente:

```
SELECT Orders.OrderId, Orders.ShipName,
Details.ProductId, Details.UnitPrice
FROM Orders
INNER JOIN [Order Details] AS Details
    ON Details.OrderId = Orders.OrderId
WHERE Details. ProductId IN
        (
            SELECT ProductId
            FROM [Order Details] AS DetSub
            WHERE DetSub. UnitPrice >= 30
        )
```

In questo caso la subquery restituisce un gruppo di righe che rispettino la condizione e pertanto, per ciascun Ordine, nel resultset complessivo della query principale, ci saranno più righe, cioè tutte quelle che hanno prodotti con prezzo unitario maggiore o uguale a 30.

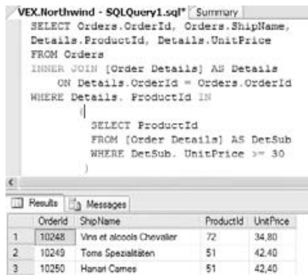


Figura 3.21: Una subquery che ritorna un resultset

Le subquery possono essere anche essere nidificate ad un numero indeterminato di livelli estendendo in modo incredibile la capacità e la profondità delle interrogazioni. Osserviamo il seguente esempio che, partendo dalla query precedente, la modifica per ritornare i soli prodotti che, oltre ad avere un valore unitario maggiore o uguale a 30, appartengono alle categorie merceologiche 1 o 3. E per far questo viene invocata un'ulteriore subquery nella tabella Products correlandola con il campo ProductId:

```
SELECT Orders.OrderId, Orders.ShipName,
Details.ProductId, Details.UnitPrice
FROM Orders
INNER JOIN [Order Details] AS Details
      ON Details.OrderId = Orders.OrderId
WHERE Details. ProductId IN
        (
            SELECT ProductId
```

```
        FROM [Order Details] AS DetSub
        WHERE DetSub. UnitPrice >= 30
            AND ProductId IN
            (
                SELECT ProductId
                FROM Products
                WHERE Products.ProductId = DetSub.ProductId
                    AND CategoryID IN (1, 2)
            )
    )
```

Figura 3.22: Subquery nidificate

3.5.5 CTE (Common Table Expression)

SQL Server 2005 introduce il supporto alle Common Table Expression, cioè alla possibilità di incapsulare blocchi di query in costrutti che possono essere riutilizzati. Considerando la query di esempio precedente, possiamo osservare come le due subquery nidificate presenti nella WHERE della query principale siano praticamente un blocco atomico, indipendente da resto e che quindi potrebbe essere logicamente tenuta in un blocco separato. Ed è quello che faremo. Osserviamo il seguente esempio:

```
WITH RigheCTE (OrderId, ProductId, UnitPrice)
```

```
AS
(
    SELECT OrderId, ProductId, UnitPrice
    FROM [Order Details] AS DetSub
    WHERE DetSub. UnitPrice >= 30
      AND ProductId IN
      (
        SELECT ProductId
        FROM Products
        WHERE Products.ProductId = DetSub.ProductId
            AND CategoryID IN (1, 2)
      )
)
```

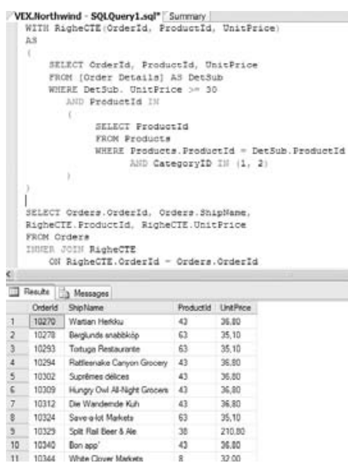


Figura 3.23: La potenza delle Common Table Expression

Il nostro CTE di esempio si chiama RigheCTE ed ecco come possiamo adoperarlo in una nuova versione dalla sintassi semplificata della query principale:

```
SELECT Orders.OrderId, Orders.ShipName,
RigheCTE.ProductId, RigheCTE.UnitPrice
FROM Orders
INNER JOIN RigheCTE
    ON RigheCTE.OrderId = Orders.OrderId
```

Si tratta sicuramente di una versione più semplice e lineare che, con alcune piccolo modifiche sintattiche può persino essere utilizzata per realizzare query ricorsive. Esaminaimo un classico esempio di CTE ricorsiva di SQL Server 2005 tratta dalla documentazione ufficiale e basata sul database AdventureWorks:

```
WITH DirectReports (ManagerID, EmployeeID, Title, DeptID, Level)
AS
(
-- definizione principale
  SELECT e.ManagerID, e.EmployeeID, e.Title, edh.DepartmentID,
    0 AS Level
  FROM HumanResources.Employee AS e
  INNER JOIN HumanResources.EmployeeDepartmentHistory AS edh
    ON e.EmployeeID = edh.EmployeeID AND edh.EndDate IS NULL
  WHERE ManagerID IS NULL
  UNION ALL
-- definizione ricorsiva
  SELECT e.ManagerID, e.EmployeeID, e.Title, edh.DepartmentID,
    Level + 1
  FROM HumanResources.Employee AS e
  INNER JOIN HumanResources.EmployeeDepartmentHistory AS edh
    ON e.EmployeeID = edh.EmployeeID AND edh.EndDate IS NULL
  INNER JOIN DirectReports AS d
    ON e.ManagerID = d.EmployeeID
)
```

PuntoInformatico Libri

```
SELECT ManagerID, EmployeeID, Title, Level
FROM DirectReports
INNER JOIN HumanResources.Department AS dp
    ON DirectReports.DeptID = dp.DepartmentID
WHERE dp.GroupName = N'Research and Development' OR Level = 0
```

Questa volta la CTE si compone di una UNION in cui la prima parte è una query normale e quindi non dissimile da quella delle CTE non ricorsive, ma la secona parte della UNION è invece costituita da una query in cui è presente in JOIN la CTE stessa che, dunque, viene invocata ricorsivamente.

PROGRAMMARE SQL SERVER IN T-SQL

Fino a questo punto del corso abbiamo trattato quella parte di SQL che assolve alle funzioni di query ovvero interrogazioni dei dati senza nessuna possibilità di manipolarli, magari cambiandone il valore. Adesso vedremo invece quella parte di SQL che assolve alle funzioni di DML (Data Manipulation Language). Questa parte di SQL ci consente di inserire dati nelle tabelle, di modificarli e di cancellarli; le corrispondenti istruzioni che assolvono a tale scopo sono: INSERT, UPDATE, DELETE. Infine osserveremo le estensioni DDL (Data Definition Language), che ci permettono di creare database, tabelle, viste, stored procedure ed ogni altro oggetto del database.

4.1 INSERIMENTO DI DATI NEL DATABASE

Possiamo operare con le tre istruzioni INSERT, UPDATE, DELETE sui dati in modo selettivo o in modo globale. Nel modo selettivo useremo delle select o delle condizioni per far riferimento a particolari valori o a particolari posizioni nella tabella, nel modo globale non faremo uso di select o di condizioni.

4.1.1 INSERT

Abbiammo già visto in Figura 3.16 la struttura delle tabelle Customers e Suppliers di Northwind e in tutto il corso del capitolo abbiamo eseguito tutte le possibili query, a titolo esemplificativo, su queste ed altre tabelle. È finalmente giunto il momento di inserire nuovi record in queste tabelle. Infatti il linguaggio SQL consente, attraverso le sue estensioni DML sopra citate, di effettuare questa operazione.

Proviamo, dunque, ad inserire un record nella tabella Customers:

```
INSERT INTO Customers
(CustomerId, CompanyName, ContactName, ContactTitle, Address, City,
                                                          Country)
```


PuntoInformatico Libri

```
VALUES
('VV1', 'Edizioni Master S.p.A.', 'Vito Vessia', 'Collaboratore', 'Via Della
                                        Collaborazione, 15', 'Bari', 'Italy')
```

La sintassi, dopo la parola chiave INSERT INTO <nome_tabella>, permette di specificare l'elenco dei campi della tabella che si intende valorizzatore e, con la clasuola VALUES, si possono indicare i corrispondenti valori. All'esecuzione del comando, se non si sono verificati errori sintattici, SQL Server 2005 segnalerà un tranquilizzante:

```
(1 row(s) affected)
```

Ad indicare proprio la riuscita dell'inserimento del nuovo record. È fondamentale che ogni riga inserita rispecchi le regole sintattiche e semantiche della tabella. Infatti è, ad esempio, possibile omettere la valorizzazione di alcuni campi della tabella purché questi accettino valori NULL, diversamente verrà indicato un errore come il seguente:

```
Cannot insert the value NULL into column 'CustomerID', table
    'Northwind.dbo.Customers'; column does not allow nulls. INSERT fails.
The statement has been terminated.
```

È altresì necessario che la chiave primaria non venga violata inserendo valori duplicati, pena l'annullamento dell'insermento e la produzione del messaggio come quello di esempio che segue:

```
Violation of PRIMARY KEY constraint 'PK_Customers'. Cannot insert
                                duplicate key in object 'dbo.Customers'.
The statement has been terminated.
```

L'inserimento di nuovi valori è possibile anche non esplicitando direttamente i valori da inserire, ma recuperando ed eventualmente manipolando i valori da un'altra query. Eccone un esempio:

```
INSERT INTO Customers
(CustomerId, CompanyName, ContactName, ContactTitle, Address, City,
                                                              Country)
SELECT 'New' + CAST(SupplierId AS CHAR(3)), CompanyName,
                                                          ContactName,
ContactTitle, HomePage, City, Country
FROM Suppliers
WHERE SupplierId IN (2, 3)
```

In questo caso viene eseguita una SELECT su Suppliers per restituire le righe che rispettano la condizione SuppliersId IN (2, 3). I campi SupplierId, CompanyName, ContactName, ContactTitle, HomePage, City, Country delle righe recuperate andranno rispettivamente a valorizzare i campi CustomerId, CompanyName, ContactName, ContactTitle, Address, City, Country. Il campo SupplierId subirà una leggera manipolazione e complessivamente queste righe diventeranno due nuove righe della tabella Suppliers, infatti SQL Server risponderà con un laconico:

```
(2 row(s) affected)
```

Se nella tabella in cui effettuare l'inserimento ci fossero campi Identity, la loro valorizzazione esplicita non sarebbe neceessaria ma di questo si occuperebbe la gestione automatica di SQL Server, attribuendo alle nuove righe dei valori incrementali del contatore.

4.1.2 UPDATE

Questa istruzione serve per modificare i dati contenuti in una tabella. Proviamo a modificare alcuni record della nostra solita tabella Northwind Customers di esempio modificando le due righe sopra inserite (la cui chiave CustomerId inizia per New), per aggiungere al campo ContactName il prefisso 'Update':

```
UPDATE Customers
```

```
SET ContactName = 'Il bravo collaboratore', ContactTitle = 'Varie'
WHERE CustomerId LIKE 'New%'
```

Anche in questo caso ci segnalerà, in caso di esito positivo, il numero di righe interessate dall'aggiornamento di valore. I campi aggornati sono ContactName e ContactTitle. È possibile anche in questo caso effettuare l'aggiornamento di campi provenienti da una query:

```
UPDATE Customers
SET ContactName = 'Il bravo collaboratore', ContactTitle = 'Varie'
FROM Customers
INNER JOIN Orders
      ON Orders.CustomerId = Customers.CustomerId
WHERE Orders.OrderId > 11076
```

Quando si effettua l'UPDATE su un resultset che implica la correlazione di più tabelle, è importante tener conto che l'aggiornamento può interessare una sola delle tabelle coinvolta dalla JOIN ed è proprio quella indicata dalla clausola UPDATE.

4.1.3 DELETE

Oltre a inserire dati in una tabella o modificarli, occorre anche poterli cancellare. Questa istruzione assolve allo scopo. Va fatto notare che DELETE non cancella un singolo campo per volta ma una riga o più righe per volta; inoltre questa istruzione cancella i record e non l'intera tabella. Vediamo ora alcuni esempi per meglio comprendere come funziona DELETE:

```
DELETE FROM Customers
WHERE CustomerId = 'VV1';
```

In questo caso vengono cancellate le righe di Customers il cui valore di CustomerId è uguale a VV1. La cancellazione, però, può interessare anche le righe dell'intera tabella incondizionatamente:

```
DELETE FROM Customers
```

Esiste però una soluzione più rapida ed efficace per svuotare l'intero contenuto della tabella costituita dall'instruzione TRUNCATE:

```
TRUNCATE TABLE Customers
```

Anche la DELETE può agire su un resultset, in tal caso, dopo la parola chiave DELETE e prima della FROM va indicato il nome della tabella su cui effettuare la cancellazione.

4.2 GESTIONE DELLE TRANSAZIONI

Quando si inseriscono o si cancellano o si modificano i dati in un database è possibile intervenire al fine di annullare l'operazione o confermarla definitivamente. Ciò è particolarmente utile quando ci accorgiamo di aver eseguito uno dei tre comandi, visti in questo capitolo, sui dati errati, o quando vogliamo confermare definitivamente il comando mandato in esecuzione.
Per far maggiore chiarezza a quanto affermato va detto che i DBMS i quali implementano i comandi ROLLBACK e COMMIT, non rendono effettivi istantaneamente i comandi DELETE, UPDATE e INSERT, ma tengono memoria temporaneamente delle modifiche effettuate in un'altra area. Questo fa si che un utente, che non sia quello che ha eseguito uno dei comandi DELETE, UPDATE e INSERT, non veda le modifiche appartate, o almeno che questa sia una delle modalità transazionali contemplate; mentre l'altro, quello che ha eseguito uno dei tre comandi, veda le tabelle in oggetto come se fossero state modificate definitivamente. Tutti salvataggi intermedi e non consolidati vengono temporaneamente salvatati nel database di log, corrispondente al file fisico .LDF, e vengono effettivamente salvatati nel database fisico vero e proprio solo alla fine, in caso di esito positivo. Dunque il comando di COMMIT rende definitive le modifiche apportate e il comando ROLLBACK elimina ogni modifica da queste ultime. Cerchiamo di meglio comprendere di quanto detto, provando

PuntoInformatico Libri

ad eseguire in transazione due operazioni di scrittura: se la prima andasse a buon fine e la seconda generessa un errore, in un contesto non transazionale resterebbe permanente solo la prima modifica e verrebbe annullata la seconda. Nell'ambito di una transazione, invece, anche la prima operazione verrà annullata se la seconda è andata male. Per avviare una nuova transazione viene utilizzato il comando BEGIN TRANSACTION. Una transazione può essere completata positivamente con una COMMIT TRANSACTION oppure il tutto può essere annullato con una ROLLBACK TRANSACTION. Infine è possibile effettuare delle COMMIT intermedie in modo che se un'istruzione va in errore e si è effettuato un salvataggio intermedio, la ROLLBACK si applicherà solo su quello che resta. Osserviamo il seguente esempio:

```
BEGIN TRANSACTION
--operazione "a"
INSERT INTO Customers
(CustomerId, CompanyName, ContactName, ContactTitle, Address, City,
                                                                Country)
VALUES
('VV1', 'Edizioni Master S.p.A.', 'Vito Vessia', 'Collaboratore', 'Via Della
                                       Collaborazione, 15', 'Bari', 'Italy')
SAVE TRANSACTION a
--operazione "b"
INSERT INTO Customers
(CustomerId, CompanyName, ContactName, ContactTitle, Address, City,
                                                                Country)
VALUES
('VV2', 'Edizioni Master S.p.A.', 'Vito Vessia', 'Collaboratore', 'Via Della
                                       Collaborazione, 15', 'Bari', 'Italy')
SAVE TRANSACTION b
--operazione "c"
INSERT INTO Customers
(CustomerId, CompanyName, ContactName, ContactTitle, Address, City,
```

```
                                                          Country)
VALUES
('VV2', 'Edizioni Master S.p.A.', 'Vito Vessia', 'Collaboratore', 'Via Della
                                      Collaborazione, 15', 'Bari', 'Italy')
--annullamento operazione "b"
ROLLBACK TRANSACTION b
DELETE FROM Customers
WHERE CustomerId = 'VV1';
COMMIT TRANSACTION
```

L'intera operazione si suddivide in tre blocchi (a, b e c): viene prima eseguito il blocco "a" che viene subito consolidato nel database, poi il blocco "b", poi viene eseguito il blocco "c" e viene annullato il blocco "b" ed infine si effettuta la COMMIT del tutto (e quindi solo del blocco "c" visto che il blocco "a" era stato già l'oggetto di un salvataggio intermedio e che il blocco "b" era stato annullatto.

4.2.1 Livelli di isolamento

SQL Server 2005 gestisce i seguenti livelli di isolamento nelle transazioni:

- READ UNCOMMITTED (è possibile leggere dall'esterno della transazione anche le modifiche effettuata in transazione prima che vi sia la COMMIT);
- READ COMMITTED (è impossibile leggere dall'esterno della transazione anche le modifiche effettuata in transazione prima che vi sia la COMMIT);
- REPETEABLE READ;
- SNAPSHOT;
- SERIALIZABLE (viene posto un blocco su tutti gli oggetti interessati dalla transazione in modo che nemmeno da altre transazioni sia possibile scrivere modifiche su righe delle stesse tabelle).

4.3 CREAZIONE E MODIFICA DI OGGETTI DEL DATABASE

Iniziamo una breve sequenza delle principali istruzioni T-SQL di tipo DDL per effettuare la creazione, la modifica e la cancellazione di oggetti del database.

4.3.1 Creazione, modifica e cancellazione di un database

La creazione di un database si effettua con il seguente comando CREATE DATABASE:

```
CREATE DATABASE [<nome_database>] ON  PRIMARY
( NAME = N'<nome_device_mdf>', FILENAME = N'<nome_file.mdf>' ,
                                        SIZE = <dimensione_in_kb>)
 LOG ON
( NAME = N'nome_device_ldf', FILENAME = N'<nome_file.ldf>' , SIZE =
                                                <dimensione_in_kb>)
```

Ecco un esempio:

```
CREATE DATABASE [Prova] ON  PRIMARY
( NAME = N'Prova', FILENAME = N'C:\Programmi\Microsoft SQL
 Server\MSSQL.1\MSSQL\DATA\Prova .mdf' , SIZE = 3072KB , FILEGROWTH
                                                      = 1024KB )
 LOG ON
( NAME = N'Prova_log', FILENAME = N'C:\Programmi\Microsoft SQL
        Server\MSSQL.1\MSSQL\DATA\Prova_log.ldf' , SIZE = 1024KB , FILE
                                                   GROWTH = 10%)
```

La modifica delle impostazioni di un database si effettua con il comando ALTER DATABASE. Esempio:

```
ALTER DATABASE [<nome_database>] SET <nome_proprieta>
```

```
<valore_proprietà>
```

Ecco un esempio:

```
ALTER DATABASE [Prova] SET ANSI_NULLS OFF
```

Le proprietà del database sono numerose, come abbiamo visto nel capitolo relativo all'amminstrazione e per cui si rimanda a quella sede e alla documentazione ufficiale per una disamina completa.
Infine, la cancellazione di un database si effettua col comando DROP DATABASE:

```
DROP DATABASE [<nome_database>]
```

4.3.2 Creazione, modifica e cancellazione di una tabella

La creazione di una tabella è possibile attraverso l'istruzione CREATE TABLE. Eccone la sintassi:

```
CREATE TABLE [ schema.] nome_tabella
(
    <nome_campo_1> <tipo_campo_1> [NOT NULL|NULL] [, ...n]
)
CONSTRAINT <nome_pk> PRIMARY KEY [CLUSTERED|NON CLUSTERED]
(
    <nome_campo1_pk <tipo_ordinamento> [, ... nome_campon_pk
                                              <tipo_ordinamento>] >
)
) ON [PRIMARY]
```

Osserviamo il seguente esempio di creazione della tabella Orders in Northwind:

```
CREATE TABLE [dbo].[Orders]
```


PuntoInformatico Libri

```
(
	[OrderID] [int] IDENTITY(1,1) NOT NULL,
	[CustomerID] [nchar](5) COLLATE SQL_Latin1_General_CP1_CI_AS
NULL,
	[EmployeeID] [int] NULL,
	[OrderDate] [datetime] NULL,
	[RequiredDate] [datetime] NULL,
	[ShippedDate] [datetime] NULL,
	[ShipVia] [int] NULL,
	[Freight] [money] NULL CONSTRAINT [DF_Orders_Freight]  DEFAULT
(0),
	[ShipName] [nvarchar](40) COLLATE SQL_Latin1_General_CP1_CI_AS
NULL,
	[ShipAddress] [nvarchar](60) COLLATE
SQL_Latin1_General_CP1_CI_AS NULL,
	[ShipCity] [nvarchar](15) COLLATE SQL_Latin1_General_CP1_CI_AS
NULL,
	[ShipRegion] [nvarchar](15) COLLATE
SQL_Latin1_General_CP1_CI_AS NULL,
	[ShipPostalCode] [nvarchar](10) COLLATE
SQL_Latin1_General_CP1_CI_AS NULL,
	[ShipCountry] [nvarchar](15) COLLATE
SQL_Latin1_General_CP1_CI_AS NULL,
 CONSTRAINT [PK_Orders] PRIMARY KEY CLUSTERED
(
	[OrderID] ASC
)
) ON [PRIMARY]
```

Il tipo delle colonne è tra quelli previsti da SQL Server 2005. Possiamo osservare come alcuni campi stringa indichino anche il tipo di collation. La chiave primaria deve assumere necessariamente un nome e può contenere uno o più campi della tabella stessa. Inoltre di fatto è un indice, di

default di tipo CLUSTERED. L'aggiunta di eventuali foreign key si effettua attraverso l'esecuzione del comando ALTER TABLE, come mostrato nell'esempio:

```
ALTER TABLE [dbo].[Orders]  WITH NOCHECK ADD  CONSTRAINT
                          [FK_Orders_Customers] FOREIGN KEY([CustomerID])
REFERENCES [dbo].[Customers] ([CustomerID])
```

In questo caso il campo CustomerId della tabella Orders viene messo in foreign key con il campo omonimo che è chiave primaria nella tabella Customers. In generale, però, la ALTER TABLE, con la stessa sintassi, può essere utilizzata per modificare la struttura di una tabella esistente. Ad esempio è possibile aggiungere una nuova colonna:

```
ALTER TABLE dbo.Orders ADD
     NuovaColonna datetime NULL
Oppure cancellarne una esistente:
ALTER TABLE dbo.Orders
     DROP COLUMN ShipPostalCode
```

Infine è possibile rimuovere per intero la tabella:

```
DROP TABLE Orders
```

4.3.3 Creazione, modifica e cancellazione di una vista

La creazione di una vista è possibile attraverso l'istruzione CREATE VIEW. Eccone la sintassi:

```
CREATE VIEW [ schema.] nome_vista
[ WITH <attributi> [ ,...n ] ]
AS
<query>
```

PuntoInformatico | Libri

```
[ WITH CHECK OPTION ]
```

Dunque si esprime un nome della vista eventualmente corredato dallo schema di appartenenza, se gestito, possono essere specificati uno o più attributi (che possono essere ENCRYPTION, SCHEMABINDING e VIEW_METADATA e per la cui comprensione si rimanda alla documentazione ufficiale del produttore); a questo punto viene riportata la query SELECT vera e propria. Essa può contenere tutte le clausole ad accezione di COMPUTE, COMPUTE BY, ORDER BY (a meno che non vi sia anche una TOP nella clasuola SELECT), OPTION ed INTO.
Osserviamo il seguente esempio tratto da Northwind:

```
CREATE VIEW [dbo].[Products by Category]
AS
SELECT Categories.CategoryName, Products.ProductName,
     Products.QuantityPerUnit, Products.UnitsInStock, Products.Discontinued
FROM Categories INNER JOIN Products ON Categories.CategoryID =
                                                        Products.CategoryID
WHERE Products.Discontinued <> 1
WITH CHECK OPTION
```

Ed ecco come è possibile utilizzare la vista in una query come se fosse una tabella:

```
SELECT *
FROM [Products by Category]
WHERE UnitsInStock > 20
```

Il contenuto delle viste può essere modificato con INSERT, UPDATE e DELETE purché queste agiscano esclusivamente su una sola delle tabelle eventualmente in JOIN nella query interna della vista. Con la WITH CHECK OPTION, opzionale, infine è possibile esprimere l'obbligo secondo il quale tutte queste operazioni di modifica agiscano sempre e solo in rispetto

alla condizione di filtro della vista. Osservando la vista di esempio precedente, si nota che vengono restituite le sole righe che rispecchiano la condizione Products.Discontinued <> 1. Se provassimo ad eseguire la seguente UPDATE che modifica tale condizione per un record:

```
UPDATE [Products by Category]
SET Discontinued = 1
WHERE ProductName = 'Ravioli Angelo'
```

Il database engine ci restituirebbe il seguente risultato:

```
The attempted insert or update failed because the target view either speci
fies WITH CHECK OPTION or spans a view that specifies WITH CHECK
OPTION and one or more rows resulting from the operation did not qua
lify under the CHECK OPTION constraint.
The statement has been terminated.
```

Le viste sono modificabili con la ALTER VIEW che ha la medesima sintassi della CREATE VIEW ma agisce su viste già esistenti. Infine sono cancellabili con:

```
DROP VIEW [ schema.] nome_vista
```

4.3.4 Creazione, modifica e cancellazione di un indice

La creazione di un indice su una tabella o su una vista (quest'ultima è una nuova funzionalità introdotta da SQL Server 2005) è possibile attraverso l'istruzione CREATE INDEX. Osserviamone la sintassi semplificata da cui sono state omesse alcune parti non di uso comune:

```
CREATE [ UNIQUE ] [ CLUSTERED | NONCLUSTERED ] INDEX
<nome_indice>
ON <[schema.].nome_tabella_o_vista>
```

PuntoInformatico Libri

```
( colonna [ ASC | DESC ] [ ,...n ] )
   [ INCLUDE ( colonna [ ,...n ] ) ]
```

Innanzitutto la presenza della parola chiave opzionale UNIQUE indica che l'indice che si andrà a creare è univoco e pertanto, nell'ambito della tupla di campi che si andrano ad indicizzare non potranno esservi record con duplicazioni, rendendo così di fatto un indice UNIQUE una chiave primaria alternativa o surrogata. Normalmente gli indici rappresentano un ordinamento alternativo alla posizione dei record nella tabella fisica, ed è questo il comportamento di default quando si crea un indice omettendo l'opzione CLUSTERED/ NONCLUSTERED o indicando l'opzione NON CLUSTERED. La parola chiave CLUSTERED, invece, forza l'ordinamento indicato dall'indice a livello di record di tabella fisica riorganizzandone tutte le posizioni. È evidente che in una tabella o vista vi può essere solo un indice CLUSTERED. Di default, l'indice associato alla chiave primaria di ciascun tabella è di tipo CLUSTERED, ma questo comportamento è modificabile. È dunque fondamentale tener presente il notevole dispendio di risorse necessario a generare o modificare un indice CLUSTERED su una tabella già popolata e di notevoli dimensioni.
Ciascuna delle colonne indicizzate può essere ordinata in modalità crescente o decrescente. Infine SQL Server 2005 introduce due importanti novità sugli indici: la possibilità di indicizzare anche le viste, opzione che colma il gap prestazionale rispetto alle tabelle fisiche e la possibilità di aggiungere delle colonne aggiuntive (nella clausola INCLUDE) che consentono di aggiungere nell'indice anche altri campi della tabella fisica o della vista, evitando così di dovervi accedere per leggere il valore di tali campi, una volta individuato il record attraverso l'indice. Tali campi aggiuntivi, però, non concorrono alla geometria dell'indice ma sono da considerarsi esclusivamente informazioni aggiunte. SQL Server 2005 utilizza automaticamente questi campi in query che fanno uso di indici e che accedono ai campi estensivi previsti dall'indice, analogamente a quanto avviene per i campi indicizzati veri e proprio. La INCLUDE è utilizzabile solo su indici NONCLUSTERED.

Vediamo un esempio di creazione di un indice nel database di esempio Pubs:

```
CREATE NONCLUSTERED INDEX [employee_new] ON [dbo].[employee]
(
    [lname] ASC,
    [fname] DESC,
    [minit] ASC
)
INCLUDE
(
    job_lvl, job_id
)
```

La rimozione di un indice è possibile attreverso la DROP INDEX:

```
DROP INDEX <nome_indice> ON <[schema.].nome_tabella_o_vista>
```

PuntoInformatico Libri

NOTE

NOTE

PuntoInformatico | Libri

NOTE

NOTE

PuntoInformatico | Libri

NOTE

PuntoInformatico | Libri

i libri di

ioPROGRAMMO

LAVORARE CON SQL SERVER

Autore: Vito Vessia

EDITORE
Edizioni Master S.p.A.
Sede di Milano: Via Ariberto, 24 - 20123 Milano
Sede di Rende: C.da Lecco, zona ind. - 87036 Rende (CS)

Realizzazione grafica:
Cromatika Srl
C.da Lecco, zona ind. - 87036 Rende (CS)

Art Director: Paolo Cristiano
Responsabile grafico di progetto: Salvatore Vuono
Coordinatore tecnico: Giancarlo Sicilia
Illustrazioni: Tonino Intieri
Impaginazione elettronica: Francesco Cospite

Servizio Clienti
☎ **Tel. 02 831212 - Fax 02 83121206**
@ **e-mail: customercare@edmaster.it**

Stampa: Grafica Editoriale Printing - Bologna

Finito di stampare nel mese di Giugno 2007

Il contenuto di quest'opera, anche se curato con scrupolosa attenzione, non può comportare specifiche responsabilità per involontari errori, inesattezze o uso scorretto. L'editore non si assume alcuna responsabilità per danni diretti o indiretti causati dall'utilizzo delle informazioni contenute nella presente opera. Nomi e marchi protetti sono citati senza indicare i relativi brevetti. Nessuna parte del testo può essere in alcun modo riprodotta senza autorizzazione scritta della Edizioni Master.

Copyright © 2007 Edizioni Master S.p.A.
Tutti i diritti sono riservati.