I NETBOOK
trovati in rete

UN MANUALE INTRODUTTIVO PER APPRENDERE LE BASI
DEL RDBMS DISTRIBUITO DA MICROSOFT

INTRODUZIONE A SQL SERVER

Dott. Maurizio Boghetto

http://www.bomasoftware.it/download_utility/guide/SQLServer.pdf

Introduzione a SQL Server

Dott. Maurizio Boghetto

Indice

1 I Database Relazionali
2 Il Modello Relazionale
3 Structured Query Language e linguaggio Transact
4 Trigger e stored procedures

Capitolo primo

I Database Relazionali

---

1. Cos'è un database relazionale.

Microsoft Access, Oracle, Informix, Microsoft SqlServer sono sicuramente nomi che comunemente vengono associati alla gestione di archivi informatici. Ma cosa rappresentano tutte queste sigle e cosa hanno in comune ?
Tutti quanti sono RDBMS (Relational DataBase Management System) ovvero un insieme di software che si occupano di gestire le basi di dati ed in particolare le basi di dati Relazionali.
Tutti quanti, inoltre, comunicano per mezzo di un linguaggio detto SQL (Structured Query Language) che è il linguaggio standard delle basi di dati relazionali[1].

Ma cosa sono le Basi di Dati ?

E' difficile avere una concezione intuitiva delle Basi di Dati.
Al di là di quelle che possono essere le definizioni formali delle basi di dati, proviamo a fare questo ragionamento:
Un testo elettronico (per intenderci un documento di Microsoft Word) può essere intuitivamente considerato come un comune **un foglio a righe.**
Un foglio di calcolo (per intenderci un foglio Microsoft Excel) può essere intuitivamente pensato come un **foglio a quadretti** su cui fare dei calcoli, annotare dei dati e così via.
Ma il database sfugge a questa rappresentazione intuitiva.

Si può definire il database come:

UNA RAPPRESENTAZIONE DELLA REALTÀ.

E' certamente una realtà di nostro interesse, come può essere il magazzino di una impresa, la biblioteca di una scuola, la prenotazione di una stanza d'albergo.
Chi fa il database deve conoscere esattamente queste realtà che riprodurrà nella struttura da realizzare.
Non si tratta quindi di annotare i propri dati su di un foglio per catalogarli. Per questo ci sono i fogli di calcolo che su di una grande "tabellona" o su più tabellone li immagazzinano.

[1] Dopo aver definito cosa queste sigle hanno in comune occorre per correttezza anche dire le differenze. Mentre Access è un RDBMS client Oracle, SQLServer, Informix sono RDBMS server. Inoltre, ognuno di loro ha sviluppato un proprio dialetto del linguaggio SQL e la comunicazione avviene tramite dei "traduttori" del dialetto in SQL standard.

Si dovrà realizzare una struttura composta da un certo numero di tabelle collegate fra di loro da legami logici che siano l'espressione della realtà che vogliamo codificare.
Il modello attraverso il quale rappresentare questa realtà, e che si è affermato col tempo, è quello **Relazionale.** Per cui si stabiliscono dei legami tra le varie tabelle.

Ma perché usare un Database relazionale?
I motivi sono tanti, fra questi si segnala:

- **MIGLIORE GESTIONE DELL'INSERIMENTO, MODIFICA E CANCELLAZIONE DEI DATI.**
- **RIDUZIONE DELLA DUPLICAZIONE DEI DATI.**
- **POSSIBILITÀ DI INTERROGARE I DATI .**

Si pensi a quanto sia difficile gestire i propri dati inserendoli, modificandoli ed eliminandoli in un foglio di calcolo quando cominciano ad essere molti.
Realizzare una o più tabelle non correlate porta poi a duplicare i dati con un aumento sia dello spazio utilizzato sia del tempo richiesto per l'inserimento.
Un altra caratteristica che rende il database (e quello relazionale in particolare) estremamente utile è la sua capacità di interrogare i dati sfruttando le relazioni esistenti fra essi al fine di ricavare informazioni utili a gestire la propria attività.
Occorre anche dire perché i database relazionali si sono affermati come struttura più diffusa nel mercato mondiale.
Le regole alla base dei database relazionali arrivano direttamente dalla teoria insiemistica. Per cui queste regole sono certe e trasparenti. Non sono strutture di basi di dati (gerarchico, ad oggetti ecc..) "proprietarie" , ovvero diverse da produttore a produttore. Le fondamenta di tutti i database relazionali sono comuni e poggiano su basi scientifiche a tutti note.
Su questa base è stato realizzato un linguaggio, l'SQL appunto, che è divenuto uno standard per la creazione e soprattutto la interrogazione delle basi di dati relazionali. Investire nelle basi di dati relazionali permette quindi di possedere oltre che di una struttura solida anche di una struttura trasparente e non dipendente da un particolare produttore che può essere trasportata da una piattaforma all'altra con estrema facilità.

2. Il Modello dei Dati

Un modello di dati è uno strumento concettuale che consente al progettista di attribuire un certo significato (o interpretazione) ai dati e di manipolare i dati stessi. Si assegna una struttura ai dati attraverso appositi meccanismi di strutturazione previsti dal modello[2]*.*
Il dato non dà di per se informazioni se non gli si fornisce una chiave d'interpretazione. Il modello di dati è uno strumento concettuale tramite il quale si può acquisire conoscenza da un insieme di dati altrimenti insignificante. L'organizzazione dei dati all'interno di questo schema attribuisce al dato una interpretazione che nel caso del modello relazionale viene definita come **rappresentazione della realtà di interesse** (Codd).

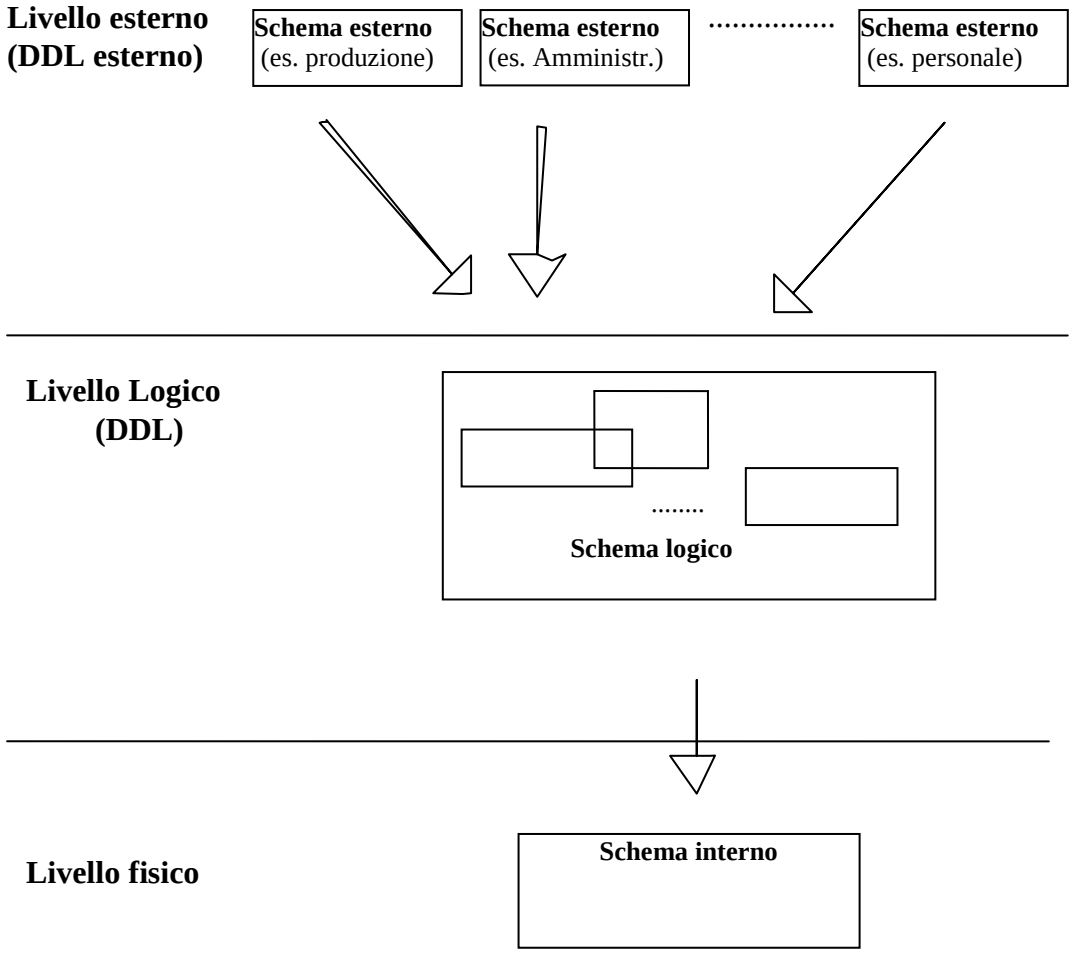
1. Indipendenza fisica e logica e i livelli di descrizione dei dati

Un requisito irrinunciabile dei DBMS è che devono garantire le proprietà d'*indipendenza fisica* e *logica*.
Per indipendenza (delle applicazioni dall'organizzazione) fisica (dei dati), s'intende la possibilità che i programmi applicativi non debbano essere modificati in seguito a modifiche dell'organizzazione fisica dei dati[3]. Le ragioni principali delle modifiche dell'organizzazione fisica dei dati possono essere :

- Un'evoluzione dell'uso dei dati può rendere necessario dover intervenire sull'organizzazione fisica della base dei dati.
- Cambiano i dispositivi fisici di memorizzazione, le tecniche di compattamento o di trasmissione dati, il sistema operativo oppure il suo sottosistema per la gestione degli archivi.
- Se la base di dati è distribuita su più nodi di una rete il DBMS può cambiare la locazione dei dati memorizzati oppure il tipo di calcolatore su cui sono memorizzati.

[2] Corso di informatica generale - G. Callegaris CEDAM
Un'altra definizione è quella di Albano :"Un modello dei dati è un insieme di meccanismi di strutturazione, o di astrazione, per modellare una base di dati, con certi operatori e vincoli di integrità predefiniti"
[3] Albano, Orsini- Basi di dati Boringhieri 1985

Per indipendenza (delle applicazioni dall'organizzazione) logica (dei dati), s'intende la possibilità che i programmi applicativi non devono essere modificati in seguito a modifiche dello schema logico, per aggiunte di nuove definizioni oppure per modifiche o eliminazione di definizioni già esistenti[4].
Per soddisfare questi requisiti, è stato proposto che i DBMS offrano tre livelli distinti di descrizione dei dati : *Schema logico, interno* ed *esterno*[5].



Architettura a tre livelli di un DMBS

Lo schema logico descrive la struttura della base di dati non facendo nessun riferimento alla sua organizzazione fisica o al modo in cui vengono memorizzati i dati nelle memorie secondarie.
Lo schema interno è la descrizione di come sono organizzati fisicamente i dati nelle memorie di massa, e di quali strutture dati ausiliarie sono previste per facilitarne l'uso.
Lo schema esterno è una descrizione di come appare la struttura della base di dati ad una certa applicazione. Questo schema definisce quelle che si chiamano le viste, ovvero quelle porzioni della base di dati cui l'utente direttamente o tramite le applicazioni può accedere. In generale esistono più schemi esterni, uno per ogni gruppo omogeneo d'utenti. Essi però non sono indipendenti, poiché gli oggetti in comune hanno una rappresentazione unica nella base di dati e quindi le modifiche si riflettono su tutti gli utenti che ne fanno uso.
Fra questi livelli di descrizione dei dati devono esistere delle corrispondenze che vengono utilizzate dai DBMS per convertire le operazioni sugli oggetti virtuali accessibili da uno schema esterno in quelle sugli oggetti dello schema logico e, quindi, sui dati realmente presenti nel sistema, memorizzati

[4] Albano, Orsini- Basi di dati Boringhieri 1985
[5] Questo approccio è stato proposto dal comitato ANSI/X3/SPARC (1985).

secondo lo schema interno. Questi schemi sono gestiti dal DBMS e non dai programmi applicativi i quali, per accedere alla base di dati, comunicano al sistema con particolari accorgimenti a quale schema fanno riferimento.
Con questo sistema dei tre livelli di descrizione dei dati si garantisce sia l'indipendenza fisica, perché i programmi applicativi fanno riferimento allo schema esterno che non contiene informazioni sul modo in cui i dati sono organizzati fisicamente, sia l'indipendenza logica, perché i programmi fanno riferimento allo schema esterno e non allo schema logico.

2. Indirizzamento tramite indici

I gestori di database usano la tecnica degli indici per associare ad un dato presente nell'archivio un **puntatore** o **indice** che indica dove il dato **fisicamente** si trova. Lo scopo principale è quello di velocizzare l'accesso ai dati dell'archivio e quindi di velocizzare anche un'interrogazione dei dati (query) comportando però dei carichi aggiuntivi alla gestione dei dati. Infatti, quando vengono inseriti nuovi record il DBMS dovrà aggiornare anche gli archivi indice ad essi associati causando un incremento dei tempi d'esecuzione. Per questo motivo l'Amministratore di database dovrà valutare quanto incide l'operazione d'aggiornamento sulle applicazioni rispetto a quella di interrogazione. In questi casi si può far riferimento a due tipi di file : file dati e i file indice.
I file dati sono costituiti dalle informazioni della propria base di dati. Tipicamente, un file dati ha l'aspetto di una serie di record di lunghezza fissa composti da campi. Alcuni campi all'interno del record sono unici per quel record e specificano il percorso con il quale si accederà al record stesso, cioè si tratta solitamente delle chiavi primarie. Questi campi identificativi dei record vengono ripresi in un file separato denominato **file indice**. All'interno di un file indice le chiavi sono disposte secondo una di più strutture che permettono di accedere velocemente ai dati e di eseguire rapidamente inserimenti, cancellazioni e recuperi. Quasi tutti i sistemi di gestione delle basi di dati offrono più modi d'accesso fisico ai dati . I vari tipi d'operazione che si vogliono svolgere e la natura dei dati determinano quale modo o quali metodi diano le prestazioni migliori.

5. Progettare un Database (progettazione concettuale)

Per progettare un Database occorre:

Aver chiaro lo scopo per cui si vuole costruire la Base di Dati.

Conoscere la realtà di interesse.

E' logico che occorre trarre un effettivo beneficio dalla realizzazione di una base di dati per gestire efficacemente la propria realtà di interesse.
Per conoscere la realtà di interesse occorre realizzare tutta una serie di attività che costituiscono la progettazione vera e propria. In particolare occorre realizzare:

- **Una analisi conoscitiva della realtà di interesse**
- **Codificare questa realtà in uno schema (progettazione concettuale)**
- **Rappresentare la realtà secondo il modello relazionale (progettazione logica).**

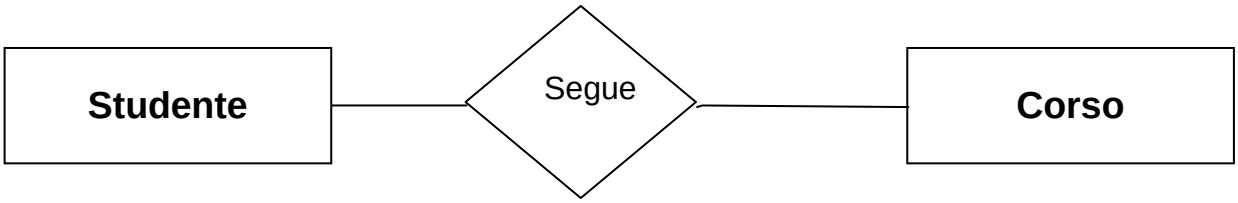
Attraverso la prima attività si deve identificare le informazioni che servono e che si dovrà inserire nel DB. Queste informazioni, poi, andranno a comporsi in uno schema rappresentativo della realtà di interesse che sarà detto schema concettuale.
Questo schema viene rappresentato graficamente per mezzo del modello

Entità-Relazioni.

Gli elementi essenziali di questo modello sono:

- **Le entità**: L'entità è un "qualcosa" che nel contesto in esame ha una certa importanza. Ad esempio, se si vuol studiare una realtà scolastica, un'entità descriverà lo studente, del quale si potranno considerare le singole proprietà che interessano (detti attributi), come nome cognome, facoltà cui è iscritto, esami sostenuti e punteggio. Un'altra entità potrà essere il corso caratterizzato da un suo codice, descrizione numero ore annue ecc.

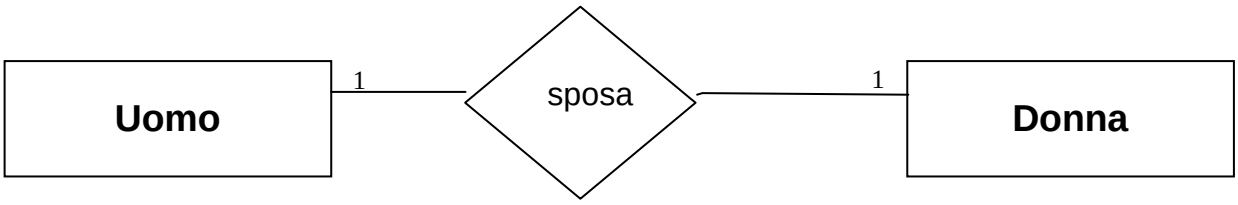
- **Gli attributi**: Con attributi si indicano dei dati che appartengono ad una entità.
- **Le Associazioni**: Indicano il tipo di relazioni fra le varie entità.
- **Le Relazioni**: Sono le relazioni fra le diverse entità. **Anche le relazioni possono avere i loro attributi**



Le relazioni possono essere di tre tipi:

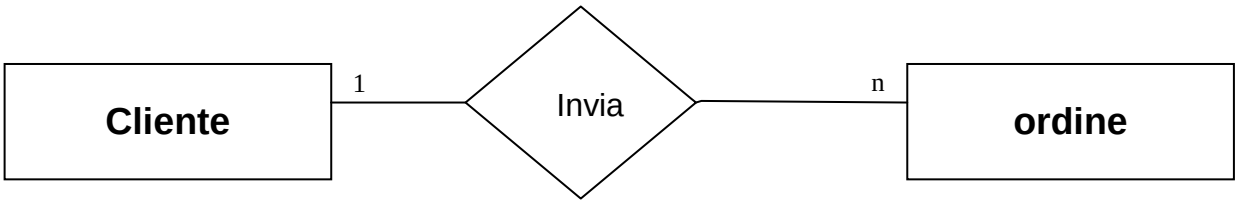
Associazioni 1 a1

Si hanno quando ad un elemento della prima entità corrisponde uno ed un solo elemento della seconda.



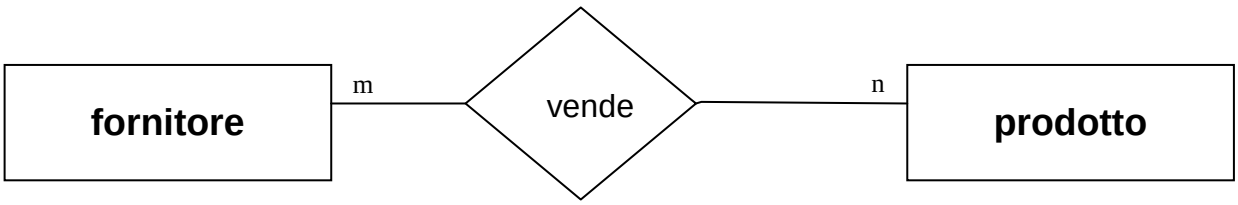
Associazioni 1 a n (oppure 1 ad infinito)

Si hanno quando ad uno ed un solo elemento di una entità corrisponde n elementi di un altra entità.



Associazioni m a n (oppure infinito ad infinito)

Si hanno quando ad un elemento di una entità A corrispondono più elementi di una entità B e viceversa.



Come si vede le entità sono rappresentate graficamente da quadrati mentre le relazioni da rombi. A volte le relazioni sono rappresentate da delle semplici linee**.**
Una volta realizzato lo schema concettuale lo si codificherà secondo le specifiche del modello relazionale.
Attraverso lo schema ER si arriva a comporre una rappresentazione della realtà oggetto del nostro interesse. Questa realtà così codificata dovrà essere trasformata secondo le regole di un modello di rappresentazione dei dati (in questo caso il modello Relazionale).
I metodi adottati per realizzare una progettazione concettuale sono solitamente l'incontro di questi due:

1. **Top-Down**. Si parte da una situazione estremamente sintetica per poi passare a gradi di analiticità maggiori fino a realizzare uno schema adeguatamente rappresentativo della nostra realtà.
2. **Bottom-Up**. Si parte da una situazione estremamente analitica che viene passo passo sintetizzata fino a realizzare uno schema adeguatamente rappresentativo della nostra realtà.

Solitamente la progettazione concettuale rappresenta un processo in cui questi due metodi si incontrano.
Maggiore è la conoscenza della realtà di interesse, maggiore sarà il grado di analiticità con cui si inizia la nostra analisi.
Minore è il grado di conoscenza della realtà di interesse, maggiore sarà il grado di sinteticità con cui si inizia la nostra analisi.

Prodotto lo schema concettuale non rimane adesso che realizzare la cosiddetta progettazione logica, ovvero occorrerà trasformare lo schema secondo le regole relazionali.

Capitolo Secondo

Il modello relazionale.

---

1. Schema Logico

La fase della progettazione concettuale dei dati porta a definire lo Schema Concettuale.
Il modello concettuale cerca di dare una struttura ai dati evitando il più possibile i dettagli realizzativi, per cui con lo Schema Concettuale si otterrà un documento di base che descrive la struttura del sistema informatico in maniera astratta, vale a dire astratta dal modello concettuale prescelto. Quindi, è un documento che guiderà i progettisti, nella fase della progettazione logica, alla realizzazione dello Schema logico secondo un particolare modello scelto. Un modello dei dati può essere definito come *un insieme di concetti, descritti tramite un preciso formalismo, il cui scopo è quello di permettere una rappresentazione ed una manipolazione delle informazioni che costituiscono il mondo della realtà*[6].
Il modello scelto per realizzare lo schema concettuale è quello relazionale elaborato da Codd nel 1970. Si tratta di un modello matematico per la descrizione dello schema logico in maniera indipendente dalla realizzazione fisica il quale trae le sue origini dalla teoria degli insiemi. Molti autori, in seguito, hanno approfondito l'argomento cercando di formalizzare i diversi problemi che l'utilizzo del modello dei dati ha suscitato, volendo rappresentare in un sistema informatico un frammento del mondo reale.

Alla base del modello relazionale si trova la relazione detta anche tabella o schema poiché questa è costituita su di una struttura di righe dette tuple e colonne dette domini della tabella.
Le testate della tabella Ai si dicono **attributi,** mentre il numero n dei domini viene detto **grado o arietà** della tabella**.** Le tabelle che hanno un solo attributo si dicono unarie, quelle con due attributi binarie, con tre ternarie e via dicendo. Il numero delle tuple si dice **cardinalità** (m+1).

Le proprietà delle relazioni possono essere così riassunte[7] :

- *I valori di ogni colonna sono fra loro omogenei.* I valori di un attributo appartengono allo stesso dominio (interi, stringhe di caratteri, ecc..). Quindi, si può affermare che gli attributi rappresentano l'uso dei domini in una determinata tabella
- *l'ordinamento delle colonne è irrilevante.* Poiché sono sempre identificate per nome e non per posizione
- *l'ordinamento delle righe è irrilevante.* Poiché queste sono identificate per contenuto e non per posizione

[6] Schiavetti :Database, Jackson 1985
[7] Atzeni, Batini, De Antonellis :Introduzione alla teoria relazionale, Masson

- *Lo schema di una tabella R(A)* è un'espressione che contiene il nome della tabella seguito dall'elenco di tutti gli attributi della tabella stessa. Può essere sia l'intestazione della tabella sia questa espressione : Prodotto (codice, descrizione, giacenza)

2. Chiavi ed attributi di una relazione

Il concetto di chiave viene elaborato nella teoria relazionale poiché le tabelle devono essere costruite in modo tale che ogni tupla deve essere distinta per mezzo di uno o più attributi. Quindi, scopo delle chiavi è identificare univocamente queste tuple. Si distingue fra superchiave, chiave candidate e chiave primaria. La **superchiave** è l'insieme degli attributi che identificano univocamente una tuple. La **chiave candidata** è in numero minimo di attributi necessari per identificare univocamente una tuple. Può essere definita come *una superchiave dalla quale possono essere eliminati attributi senza distruggere la proprietà della identificazione univoca della tuple. Per cui una chiave candidata è una superchiave con la proprietà di non ridondanza*[8].

Fra le chiavi candidate, che possono essere più di una, se ne sceglierà una che ha il minor numero di attributi e che, a parità di attributi, ha il minor numero di caratteri. Questa è la **chiave primaria** e sarà la chiave scelta fra le chiavi candidate a rappresentare univocamente una tuple.

Rispetto alla chiave della tabella si possono distinguere gli **attributi primi**, cioè quelli che fanno parte di almeno una chiave candidata, e i restanti **attributi non primi**. Infine, quegli attributi che possono costituire la chiave di altre tabelle sono detti **chiave esterna** e permettono di stabilire associazioni fra le tabelle che rappresentano diverse entità.

Però dopo tutto questo parlare è meglio che vi faccia vedere una tabella.

Cod lettore	Cognome	Nome	Indirizzo	Città	Recapito telefonico
001	Paolino	Paperino	Via di Paperinia 51	Paperopoli	0034/45789
002	Il Gallico	Asterix	Via Del Menir 64	S.t Malo	091/567989
003	Dei Paperoni	Paperone	Via Del Deposito 1	Paperopoli	5689/487451
004	Bonaparte	Napoleone	Via dei Superbi 1	Porto Azzurro	0587/984512
005	Benso	Camillo	Via Monginevro 487	Torino	011/784645
006	Il Druido	Panoramix	Via del Menir 66	S.t Malo	091/4566654
007	Bond	James	Via Degli intrepidi 7	Frascati	007/007007

Riga **Campo** **Colonna**

Notate una cosa:

Se interessa sapere la città dove abita Paperon dei Paperoni basta trovare la riga che lo riguarda ed identificare la colonna con attributo Città. Così, scorrendo la colonna città si può identificare tutti i lettori che hanno preso in prestito un libro che abitano in una certa città.

Questo è in estrema sintesi il funzionamento della ricerca dei dati all'interno di un DB Relazionale: le ricerche vengono fatte per colonne (la colonna Città) e per riga (Paperon dei Paperoni)

[8] Nazzini, Sanges, Vaccaro :Introduzione ai Data Base relazionali

3. Traduzione dei diagrammi E|R di uno schema concettuale in un modello relazionale

Si può tradurre abbastanza agevolmente gli schemi E\R secondo le strutture delle basi di dati relazionali seguendo, però, alcune regole.

Entità e gerarchie IS-A

Le **entità** degli schemi E|R vengono tradotte in altrettante tabelle. Gli attributi delle entità vengono riportati nelle colonne mentre le occorrenze delle entità saranno tradotte in tuples.

Associazioni 1 a 1 e 1 a n

Per tradurre il legame tra due entità che viene realizzato da un'associazione 1 a 1, si deve inserire tra gli attributi di almeno una delle entità la chiave esterna, cioè quell'attributo o insieme di attributi che identificano l'entità ad essa collegata. Le associazioni 1 a n sono tradotte in modo analogo. Occorre però prestare attenzione a non inserire però **attributi ripetuti**, siano chiavi esterne o attributi normali, perché possono essere causa di problemi nella gestione delle tabelle. Ad esempio, in una relazione 1 a n fra due entità, *cliente* e *numero d'ordine,* inserisco la chiave primaria della tabella cliente nella tabella ordini divenendo chiave esterna. Se facessi l'inverso, ovvero se inserissi come chiave esterna nella relazione clienti la chiave della tabella ordini, dovrei inserire per ogni cliente tutti gli ordini creando molte tuples. Le associazioni tra entità, del tipo 1 a 1 o 1 a n, riportate sui diagrammi E\R possono aver attributi propri. Questi attributi possono essere riportati in altrettante colonne di una delle tabelle in cui sono state tradotte le entità.

Associazioni n a m

Per tradurre le associazioni n a m (molti a molti) è necessario introdurre una nuova tabella. Questo procedimento, sconsigliato per le associazioni 1 a 1 e 1 a n , è la strada obbligata per questo tipo di associazione (poiché col metodo precedente si avrebbe una grande proliferazione di attributi multipli). La nuova tabella includerà gli attributi propri della associazione e le chiavi esterne che permettono i collegamenti con le due tabelle traduzione delle entità che tramite essa sono associate. La chiave primaria della nuova tabella è costituita dalla coppia primaria di chiavi esterne. Si spezza così un'associazione m a n in due associazioni 1 a n.

Articoli

FATTURA	Torrone	Spumante	Gianduia	Panettone	Panforte	Cioccolata	Panepepato	Champagne	Pandoro
01		**6**		**15**		**5**		**5**	
02						**5**	**5**		
03			**120**						
04				**10**	**30**				
05	**10**							**50**	
06	**20**					**20**			
07		**20**	**20**		**15**		**11**		**6**
08								**12**	
09		**5**	**6**			**10**			
10	**15**			**5**			**5**		

Esempio di matrice. Una relazione molti a molti si presta ad essere rappresentata sotto forma di matrice rettangolare. In questo caso si ha una matrice fra articoli e fatture. La matrice letta in verticale indica in quale fattura è stato venduto l'articolo. Letta orizzontalmente indica quali articoli sono stati venduti in una data fattura. Il numero nella casella può indicare la quantità venduta nella fattura

4. Regole di integrità

Il modello relazionale prevede due regole generali di integrità. Queste sono **la integrità di entità** e **la integrità referenziale** (o di riferimento).

Integrità di entità. Se A è un attributo della relazione R e partecipa alla chiave primaria, non può avere valori Null. Questo vuol dire che tutti gli attributi che partecipano alla chiave primaria non possono essere definiti su di un dominio che contempli valori Null.

Integrità di riferimento . Se nella relazione R la chiave primaria è rappresentata dall'attributo (o insieme di attributi) A, definito sul dominio D, allora in tutte le relazioni Sj, in cui appare come attributo A, le tuples devono contenere un valore di A definito sul dominio D oppure un valore Null. L'attributo A nelle relazioni Sj prende il nome di chiave esterna.

Per una chiave esterna va verificato se :

- Può assumere valori Null
- Cosa succede nella tabella collegata tramite chiave esterna se viene cancellato o modificato un valore della chiave primaria della tabella che ad essa si collega se questo valore è presente (nella tabella collegata) in una sua occorrenza.

La possibilità che ha una chiave esterna di assumere valori Null dipende dai tipi di Associazione. Se, ad esempio, l'associazione è quella fra Impiegato e Reparto la chiave esterna corrispondente al reparto può assumere il valore Null (un impiegato non è assegnato ad un reparto), mentre nella associazione Fattura - Movimenti la chiave esterna corrispondente alla fattura non può mai assumere il valore Null (un movimento non può esistere senza fattura).

Gli effetti dell'aggiornamento o della cancellazione di una chiave primaria, presente come chiave esterna in altre tabelle, si possono ricondurre a tre casi :

Effetto cascata. Una cancellazione o un aggiornamento della chiave primaria provoca una cancellazione o un aggiornamento delle occorrenze presenti nelle tabelle collegate tramite chiave esterna.

Effetto restrizione. La cancellazione o l'aggiornamento non devono essere permessi se sono presenti occorrenze per il valore considerato nelle tabelle collegate da chiave esterna.

Effetto annullamento. La cancellazione o l'aggiornamento di un valore della chiave primaria provoca un annullamento (Null) dei corrispondenti valori presenti nelle chiavi secondarie delle tabelle collegate da chiave esterna.

Benché il modello relazionale tramite normalizzazione crea sia l'integrità di entità sia l'ntegrità di riferimento spetterà al RDBMS (Relational DBMS) mantenere l'integrità durante il procedimento di inserimento dei dati.

In questo i diversi Rdbms differiscono molto, applicando le tecniche più disparate.

5. Operazioni relazionali

L'algebra relazionale, basata sulla teoria degli insiemi, offre una tecnica di estrazione dei dati molto efficace in un database relazionale. Per rendere possibile la manipolazione dei dati utilizza degli operatori che trasformano una o più relazioni in una nuova relazione. Possono essere individuate due categorie di operazioni :

1. *operazioni insiemistiche,* cioè si tratta di operazioni dell'algebra degli insiemi applicate alla entità relazione. Le più diffuse in ambito relazionale sono :
   - unione
   - intersezione
   - prodotto (catesiano)
   - differenza
2. *operazioni relazionali,* si tratta di operazioni che sono caratteristiche solo dell'algebra relazionale e non dell'algebra degli insiemi. Le operazioni più diffuse sono :

- selezione
- proiezione
- congiunzione (combinazione)
- divisione

Unione

L'**unione** della relazione R1(A1) con la relazione R2(A2), indicata con **R=R1∪ R2 ,** è l'insieme di tutti i tuples (righe) senza alcuna ripetizione.

R1

A_{11}	A_{12}	A_{13}
B_{11}	B_{12}	B_{13}
C_{11}	C_{12}	C_{13}

R2

K_{11}	K_{12}	K_{13}
K_{21}	K_{22}	K_{23}

R

A_{11}	A_{12}	A_{13}
B_{11}	B_{12}	B_{13}
C_{11}	C_{12}	C_{13}
K_{11}	K_{12}	K_{13}
K_{21}	K_{22}	K_{23}

Occorre, però, che vengano soddisfatte queste condizioni :

1. le due tabelle per essere unificabili devono essere **compatibili**, cioè :

Relazione di unione

di colonne)

- la k-esima colonna delle tabella R1 deve essere definita sugli stessi domini e quindi essere compatibile con la corrispondente colonna della tabella R2 (es. entrambe numeriche)

2. se esistono due righe uguali convenzionalmente ne viene indicata una sola.

3.5.1.2. Intersezione

L'**intersezione** della relazione **R1**(A1) e **R2**(A2), indicata con **R=R1** $\cap$ **R2 ,** è l'insieme di tutti i tuples o righe appartenenti sia ad **R1**(A1)che a **R2**(A2).

R1

A11	A12	A13	A14
B11	B12	B13	B14
C11	C12	C13	C14
D11	D12	D13	D14

R2

A11	A12	A13	A14
C11	C12	C13	C14
E11	E12	E13	E14

R

A11	A12	A13	A14
C11	C12	C13	C14

Relazione di intersezione

3.5.1.3. Prodotto cartesiano

Il **prodotto cartesiano** delle relazioni **R1** e **R2,** rispettivamente di grado n e k e di cardinalità m1 e m2 è la relazione indicata con **R=R1** X **R2 ,** che ha grado n+k e cardinalità m1 X m2 composta dalle tuple che si ottengono concatenando ogni tuple di **R1** con tutte le tuple di **R2**.
Per concatenare due tuples r1 (a1, a2,..., an) e r2 (b1, b2,..., bk) si costruisce una tupla che ha gli attributi di entrambe :
r1 conc. r2= (a1, a2,..., an, b1, b2,..., bk).

Si costruisce cioè con il prodotto cartesiano una tabella con lo schema R(A1,A2,...,An, B1,B2,...,Bk) le cui tuples sono formate per primi n elementi di una tupla di R1 e per i restanti k elementi da una tuple di R2.

R1

Riga R1-1
Riga R1-2
Riga R1-3
.....
.....
Riga R1-n

R2

Riga R2-1
Riga R2-2
.....
Riga R1-k

R

Riga R1-1	Riga R2-1
Riga R1-1	Riga R2-2
Riga R1-1	.....
Riga R1-1	Riga R1-k
Riga R1-2	Riga R2-1
Riga R1-2	Riga R2-2
Riga R1-2	.....
Riga R1-2	Riga R1-k
Riga R1-3	Riga R2-1
Riga R1-3	Riga R2-2
Riga R1-3	.....
Riga R1-3	Riga R1-k
.....	.....
Riga R1-n	Riga R2-1
Riga R1-n	Riga R2-2
Riga R1-n	.....
Riga R1-n	Riga R1-k

Relazione prodotto cartesiano

Differenza

La **differenza** tra relazione **R1** e **R2**, indicata con **R=R1 \ R2 (R=R1- R2) ,** è l'insieme di tutti i tuples o righe appartenenti ad **R1** ma non a **R2**[9].

R1

A_{11}	A_{12}	A_{13}	A_{14}
B_{11}	B_{12}	B_{13}	B_{14}
C_{11}	C_{12}	C_{13}	C_{14}
D_{11}	D_{12}	D_{13}	D_{14}

R2

A_{11}	A_{12}	A_{13}	A_{14}
C_{11}	C_{12}	C_{13}	C_{14}
E_{11}	E_{12}	E_{13}	E_{14}

R

B_{11}	B_{12}	B_{13}	B_{14}
D_{11}	D_{12}	D_{13}	D_{14}

Relazione differenza

Operazioni dell'algebra relazionale

Selezione

Data relazione R1 ed un predicato P semplice o composto, che opera sugli attributi di R1, ***la selezione*** *di R1 su P è una relazione R le cui tuples sono tutte le tuples di R1 che soddisfano P.*

[9] "Dati due insiemi I1 e I2, diciamo differenza I1 \ I2, l'insieme costituito dagli elementi che appartengono ad I1 che non appartengono a I2" V. Belski, Dati e basi di dati : Il modello relazionale, FrancoAngeli.

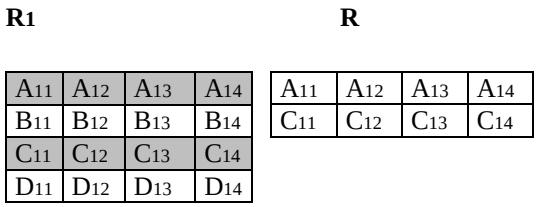


Figura 3.6 Selezione di R1 su P

Un predicato è una condizione imposta sui valori di uno o più attributi e può essere semplice oppure composto. E' semplice quando esprime una relazione del tipo :
[attributo] [operatore di confronto] [attributo]
Gli operatori di confronto sono rappresentati da : =, # (diverso), <, >, <=, >=.
E' composto se è costituito da due o più predicati collegati fra loro tramite operatori booleani AND, OR, NOT. L'operazione di selezione sceglie i tuples di una relazione in cui il predicato risulta essere vero, mentre le colonne restano le stesse. Ci si può chiedere quando un predicato composto da più predicati è vero e quindi quando verranno prelevate delle tuples. Per questo possono essere riportate le tavole di verità di Boole .

P1	P2	P1 OR P2 (P1+P2)	P1 AND P2 (P1*P2)
v	v	v	v
v	f	v	f
v	null	v	null
f	v	v	f
f	f	f	f
f	null	null	f
null	v	v	null
null	f	null	f
null	null	null	null

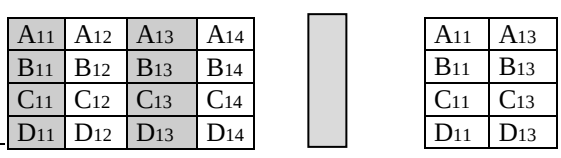
P	NOT P
V	f
f	v
null	null

Tavole di verità degli operatori booleani

Proiezione

*Data una relazione R1 di grado n ed un sottoinsieme B di k suoi attributi, (k<n) la **proiezione** di R su B è la relazione R di grado k che si ottiene da R1 ignorando le colonne degli attributi non presenti in B ed escludendo le eventuali tuple duplicate*[10]*.* Sono tuple duplicate quelle che hanno chiavi uguali. La relazione R ha la stessa cardinalità della relazione R1. L'operazione di proiezione si indica con la notazione :

B (R)



[10] V. Belski Dati e base di dati : il modello relazionale, FrancoAngeli

Figura 3.8. Proiezione di R su B

Spesso i due operatori relazionali di selezione e proiezione sono applicati insieme alla stessa relazione R1 per cui il risultato che si otterrà sarà quello di avere una relazione R, risultato delle operazioni selezione e proiezione in cui le tuple, cioè le righe, costituiscono un sottoinsieme delle tuples della relazione R1 e gli attributi, cioè le colone, costituiscono un sottoinsieme degli attributi di R1. La sequenza logica delle due operazioni prevede l'esecuzione dell'operazione di selezione e successivamente quella di proiezione.

Combinazione (giunzione, join)

Date due relazioni R1 ed R2 ed un predicato semplice P(>,<,<=, ecc.) che lega uno degli attributi di R1 con uno degli attributi di R2 con uno degli operatori booleani, la combinazione fra R1 ed R2 su P è la relazione R composta da tutte le tuple del prodotto cartesiano R1 X R2 che soddisfano P[11].
La join produce una nuova relazione mediante il procedimento seguente :

1. viene effettuato un prodotto cartesiano fra le due relazioni.
2. sulla relazione così creata viene effettuata una operazione di selezione delle tuple in cui risulta vera la condizione posta dal predicato
3. vengono ridenominati gli attributi comuni con uno stesso nome, in modo tale che compaiono una sola volta.

Se l'operatore di confronto è l'operatore di eguaglianza =, si parla allora di giunzione naturale
Join Naturale
Date due relazioni R1 (di grado n) ed R2 (di grado k) nelle quali uno degli attributi di R1 coincide con uno degli attributi di R2, Il ***join naturale*** *è la relazione r di grado (n+k-1)che contiene tutte le tuple ottenute concatenando le tuple di r1 e di r2 che presentano valori identici per l'attributo comune*[12].
Il join naturale determina una relazione tramite il seguente procedimento :

1. viene effettuato un prodotto cartesiano fra le due relazioni.
2. sulla relazione così creata viene effettuata una operazione di selezione delle tuple in cui risulta vera la condizione posta dal predicato, e cioè che gli attributi sottoposti all'operatore di confronto siano uguali. Se questa condizione non si verifica per nessun attributo allora l'operazione di giunzione si riduce a quella di un prodotto cartesiano.
3. vengono ridenominati gli attributi comuni con uno stesso nome, in modo tale che compaiono una sola volta.

Divisione

Data una relazione R1 (dividendo) di grado n e la relazione R2 (divisore) di grado k (k<n), in cui tutti gli attributi di R2 sono anche gli attributi di R1, la divisione di R1 con R2, che si indica con ***R=R1/R2,*** *è la relazione R le cui colonne sono tutte colonne di R1 che non compaiono in R2 e le cui tuple sono costituite da tutte le tuple che, concatenate con tutte le tuple di R2, danno luogo a tuple tutte presenti nella relazione R1*[13].
Si può pensare a R X R2 come ad un prodotto cartesiano sottoinsieme di R1, per cui tutte le tuple di questo prodotto cartesiano appartengono a R1 ma R1 ha altre tuple.

R1 **R2**

A_{11}	B_{11}
A_{11}	B_{12}
A_{11}	B_{13}
M_{11}	B_{11}
N_{11}	B_{12}

L'operatore di proiezione è un … e a mezzo di una lista di attributi)
con eliminazione di eventuali r… olanguages"
[11]"V. Belski Dati e base di dati
[12] V. Belski Dati e base di dati

R

B_{11}
B_{13}

A_{11}

Operazione di divisione

3.5.3. Esempi

Presidenti	**Nome**	**Partito**	**Stato**
	Eisenhower	Repubblicano	Texas
	Kennedy	Democratico	Mass.
	Johnson	Democratico	Texas
	Nixon	Repubblicano	California

1. Selezione

R1 = (select **Presidenti** where stato = "Texas")

R1	**Nome**	**Partito**	**Stato**
	Johnson	Democratico	Texas
	Eisenhower	Repubblicano	Texas

R2 = (select **Presidenti** where Partito = "Repubblicano")

R2	**Nome**	**Partito**	**Stato**
	Eisenhower	Repubblicano	Texas
	Nixon	Repubblicano	California

2. Proiezione

R3 = (project **Presidenti** over Stato, Nome)

[13] V. Belski Dati e base di dati : il modello relazionale, FrancoAngeli

R3	Nome	Stato
	Eisenhower	Texas
	Kennedy	Mass.
	Johnson	Texas
	Nixon	California

3. Unione

R4= R1∪ R2

R4	Nome	Partito	Stato
	Eisenhower	Repubblicano	Texas
	Johnson	Democratico	Texas
	Nixon	Repubblicano	California

4. Intersezione

R5 = R1∩R1

R5	Nome	Partito	Stato
	Eisenhower	Repubblicano	Texas

5. Differenza

R6 = R1-R2

R6	Nome	Partito	Stato
	Johnson	Democratico	Texas

6. Prodotto cartesiano

Studenti	N.ro Studente	Nome
	1	Paolo
	2	Claudio
	3	Nicola

Corsi	**Cod. corso**	**Descrizione**
	A	Economia
	B	Prog e Contr.
	C	Diritto trib.

Studenti X Corsi	**Num. Stud.**	**Nome**	**Cod. corso**	**Descrizione**
	1	Paolo	A	Economia
	1	Paolo	B	Prog e Contr.
	1	Paolo	C	Diritto trib.
	2	Claudio	A	Economia
	2	Claudio	B	Prog e Contr.
	2	Claudio	C	Diritto trib.
	3	Nicola	A	Economia
	3	Nicola	B	Prog e Contr.
	3	Nicola	C	Diritto trib.

7. Join (naturale)

Corsi	**Cod. corso**	**Descrizione**
	A	Economia
	B	Prog e Contr.
	C	Diritto trib.

Durata Corsi	**Cod. corso**	**Durata**
	A	4
	B	3
	C	5

$\Rightarrow$ R7 = corsi join p durata corsi

$\Rightarrow$ p = (**Corsi**. cod. corso = **Curata corsi.** cod. corso)

$\Rightarrow$ R7 = corsi. cod. corso = cod. corso. durata corsi

Corsi X Durata Corsi	**Descrizione**	**Cod. corso**	**Durata**	**Cod. corso**
	Economia	A	4	A
	Economia	A	3	B
	Economia	A	5	C
	Prog e Contr.	B	4	A
	Prog e Contr.	B	3	B
	Prog e Contr.	B	5	C
	Diritto trib.	C	4	A
	Diritto trib.	C	3	B
	Diritto trib.	C	5	C

Prodotto cartesiano **Corsi X Durata corsi**

R7	**Descrizione**	**Cod. corso**	**Durata**	**Cod. corso**
	Economia	A	4	A
	Prog e Contr.	B	3	B
	Diritto trib.	C	5	C

Selezione **Corsi.** Cod. corsi = **Durata corsi**. Cod. corsi

Dopo una operazione di proiezione per eliminare la colonna corso, avremo :

R7	Cod. corso	Durata	Descrizione
	A	4	Economia
	B	3	Prog e Contr.
	C	5	Diritto trib.

8. Divisione

Fornitori	Cod. For.	Art.
	F1	A1
	F1	A2
	F1	A3
	F2	A1
	F2	A3
	F3	A2
	F3	A1
	F4	A1
	F4	A2
	F4	A3
	F4	A4

Rich. Acquisto	Art
	A1
	A2
	A3

Quali sono i fornitori in grado di soddisfare la richiesta di acquisto ?

Fornitori :Rich. Acq.	Cod. For.
	F1
	F4

Capitolo terzo

L'SQL (Structured Query Language) e Transact

---

1. Linguaggi di interrogazione Dati

Il linguaggio SQL dialetto SQLServer (Transact SQL)

Il linguaggio SQL è un linguaggio di programmazione di database la cui origine è strettamente correlata all'invenzione del database relazionale, creato da E. F. Codd all'inizio degli anni '70. Il linguaggio SQL deriva dal linguaggio *Sequel*, nome con cui viene talvolta definito anche oggi.

Il moderno linguaggio SQL si è evoluto in uno standard largamente utilizzato per i database relazionali e definito dalle norme ANSI. Le diverse implementazioni del linguaggio SQL, compresa la versione supportata dal modulo di gestione di SQLServer, sono leggermente diverse dallo standard. Più avanti in questo capitolo verranno descritte le diverse versioni del linguaggio SQL standard, che comunque hanno in comune la stessa struttura di base e la stessa funzionalità. Chi abbia precedentemente utilizzato qualsiasi versione del linguaggio SQL non incontrerà pertanto alcuna difficoltà nel passaggio alla versione di Transact-SQL.

I Componenti del linguaggio SQL

Il linguaggio SQL è costituito da comandi, proposizioni, operatori e funzioni di aggregazione. Tali elementi vengono combinati in istruzioni che consentono di creare, aggiornare e gestire i database. Nelle sezioni che seguono verranno descritti brevemente tutti gli elementi costitutivi del linguaggio SQL, mentre nelle rimanenti parti del capitolo verranno presentati esempi specifici del loro utilizzo.

2. I Comandi

Il linguaggio SQL comprende i comandi del linguaggio di definizione dei dati (DDL) ,i comandi del linguaggio di gestione dei dati (DML) e del controllo delle transazioni e accesso (DCL). Benché in alcuni casi i due linguaggi siano sovrapponibili, i comandi DDL consentono di creare e definire nuovi database, campi e indici, mentre utilizzando i comandi DML è possibile creare query che consentono di ordinare, filtrare ed estrarre dati dal database.

DDL (Data Description Language)

Le istruzioni DDL nel linguaggio SQL sono espressioni costruite in base ai seguenti comandi.

- CREATE Consente di creare nuove tabelle, campi e indici.
- DROP Consente di eliminare tabelle e indici dal database.
- ALTER Consente di modificare tabelle aggiungendovi campi o modificandone la definizione.

DML (Data Manipulation Language)

Le istruzioni DML sono espressioni costruite in base ai seguenti comandi.

- SELECT Consente di richiedere i record del database corrispondenti a criteri specifici.
- INSERT Consente di caricare gruppi di dati nel database con una sola operazione.
- UPDATE Consente di modificare i valori di particolari record e campi.
- DELETE Consente di rimuovere record da una tabella del database.

Dcl (Data Control Language)

Istruzioni DCL che permettono di controllare l'esatta esecuzione delle operazioni o gruppi di operazioni.

- COMMIT
- ROLLBACK

Istruzioni DCL che permettono di implementare l'accesso e la sicurezza.

- GRANT
- REVOKE

3. Le Proposizioni

Le *proposizioni* sono condizioni che consentono di definire i dati che si desidera selezionare o gestire. Nella tabella che segue vengono elencate le proposizioni disponibili.

- FROM Consente di specificare il nome della tabella di cui si desidera selezionare i record.
- WHERE Consente di specificare i criteri a cui devono corrispondere i record da selezionare.
- GROUP BY Consente di suddividere in gruppi i record selezionati.
- HAVING Consente di specificare la condizione a cui deve corrispondere ciascun gruppo.
- ORDER BY Consente di ordinare i record selezionati in base all'ordine specificato.

4. Le Operazioni

Il linguaggio SQL prevede due tipi di operatori, ovvero gli operatori logici e di confronto.

Operatori logici

Gli *operatori logici* consentono di collegare due espressioni, in genere all'interno di una proposizione WHERE. Ad esempio:

```
SELECT * FROM Tabella WHERE condizione1 AND condizione2
```

In questo caso l'operatore AND collega le espressioni `condizione1` e `condizione2` per specificare che entrambe le condizioni devono essere soddisfatte per confermare il criterio di selezione. Di seguito sono elencati gli operatori logici disponibili:

- AND
- OR
- NOT

Operatori di confronto

Gli *operatori di confronto* consentono di confrontare il valore relativo di due espressioni per determinare l'azione che verrà eseguita. Ad esempio:

```
SELECT employeeid,Lastname,firstname
FROM employees where
region='WA';

employeeid  Lastname             firstname
----------- -------------------- ----------
1           Davolio              Nancy
2           Fuller               Andrew
3           Leverling            Janet
4           Peacock              Margaret
8           Callahan             Laura

(5 row(s) affected)
```

```
select lastname,birthdate
from employees
where birthdate='12/08/1948';

lastname           birthdate
-------------------- -------------------------
Davolio            1948-12-08 00:00:00.000

(1 row(s) affected)
```

In questo caso l'operatore "=" specifica che verranno selezionati solo i record il cui campo `region` contiene il valore `WA`.

Nella tabella che segue vengono elencati gli operatori di confronto:

Operatore	**Significato/utilizzo**
<	Minore di
<=	Minore o uguale a
>	Maggiore di
>=	Maggiore o uguale a
=	Uguale a
<>	Diverso da
BETWEEN	Consente di specificare un intervallo di valori
LIKE	Utilizzato per i criteri di ricerca
IN	Consente di specificare record in un database

5. Le Funzioni

Le *funzioni di aggregazione* vengono utilizzate in una proposizione SELECT e applicate a più gruppi di record per restituire un valore singolo riferito a un gruppo di record. La funzione di aggregazione AVG, ad esempio, consente di restituire la media di tutti i valori di un particolare campo di un gruppo di record. Nella tabella che segue vengono elencate le funzioni di aggregazione disponibili.

Funzione di aggregazione	
AVG	Consente di ottenere la media dei valori di un particolare campo.
COUNT	Restituisce il numero di record selezionati.
SUM	Restituisce la somma di tutti i valori di un particolare campo.
MAX	Restituisce il valore massimo del campo specificato.
MIN	Restituisce il valore minimo del campo specificato.

6. Operazioni DDL

Il linguaggio di definizione dei dati SQL include numerosi comandi che consentono di creare tabelle e indici, nonché di modificare le tabelle aggiungendovi o rimuovendo colonne o indici.

6.1 Alcuni tipi di dato

È possibile utilizzare la proprietà Tipo dati per determinare il tipo di dati memorizzati in un campo tabella. Ciascun campo è in grado di memorizzare dati di un solo tipo.

Tipo	Descrizione
Int / Integer (Integers Datatypes)	Permette di registrare numeri interi. range –2.147.483.648 to 2.147.483.647.Occupa 4 byte
Smallint (Integers Datatypes)	Permette di registrare numeri interi. range –32768 to +32767.Occupa 2 Byte
Tinyint (*floating-point* Datatypes)	Permette di registrare numeri con virgola range 3.4E–38 to 3.4E+38.Occupa 4Byte
Real **(***floating-point* Datatypes)	Permette di registrare numeri con virgola range 3.4E–38 to 3.4E+38.Occupa 4Byte
Float[(n)] **(***floating-point* Datatypes)	Permette di registrare numeri con virgola range 1.7E–308 to 1.7E+308. .Occupa 8Byte
Decimal / numeric[(p[, s])] **(***floating-point* Datatypes)	Permette di registrare numeri con virgola range 10**38–1 through –10**38 usando da 2 a 17 bytes per registrazione P=numero di cifre a sinistra e a destra della virgola S(eventuale)=Indica il numero di cifre a destra della virgola
Char (dimensione) (*character datatypes*)	Stringhe a lunghezza fissa, al massimo 255 Spazio occupato comunque
Varchar (dimensione)	Stringhe di lunghezza variabile
Datetime (datetime and *smalldatetime* Datatypes)	Si possono registrare date da 1/1/1753 AD a 12/31/9999 AD.Occupa 8 bytes.Include Data Ora
Smalldatetime (datetime and *smalldatetime* Datatypes)	Si possono registrare date da 1/1/1900 AD a 6/6/2079 .Occupa 4 Byte
Bit (Specialized Datatypes)	Dato usato per registrare situazioni con due stati 0 oppure 1
Timestamp (Specialized Datatypes)	Campo contatore
Binary(n) (Specialized Datatypes)	Registra binario lungo fino a 255 byte
Varbinary(n) (Specialized Datatypes)	Registra binario lungo oltre 255 byte
Text (text and *image* Datatypes)	Grandi testi da 1 a 2.147.483.647 bytes
Image (text and *image* Datatypes)	Binario lungo da 1 a 2.147.483.647 bytes
Money	Tipo di dato numerico. Il range è tra –922.337.203.685.477,5808 a 922.337.203.685.477,5807
Smallmoney	Tipo di dato numerico. Il range è tra –214.748,3648 to 214.748,3647
Sysname	E' un Varchar (30) che non ammette valore null.E' usato per definire le colonne in un sistema di tavole.

Creare un Database

Crea un nuovo database e i file utilizzati per archiviare il database oppure collega un database dai file di un database creato in precedenza.

Sintassi

CREATE DATABASE *database_name*
[ON
[< filespec > [**,**...*n*]]
[**,** < filegroup > [**,**...*n*]]
]
[LOG ON { < filespec > [**,**...*n*] }]
[COLLATE *collation_name*]
[FOR LOAD | FOR ATTACH]
< filespec > ::=
[PRIMARY]
([NAME **=** *logical_file_name* **,**]
FILENAME **=** '*os_file_name*'
[**,** SIZE **=** *size*]
[**,** MAXSIZE **=** { *max_size* | UNLIMITED }]
[**,** FILEGROWTH **=** *growth_increment*] **)** [**,**...*n*]
< filegroup > ::=
FILEGROUP *filegroup_name* < filespec > [**,**...*n*]

Argomenti

database_name
Nome del nuovo database. I nomi di database devono essere univoci all'interno di un server e conformi alle regole per gli identificatori. Il numero massimo di caratteri consentito per l'argomento *database_name* è 128, a meno che per il file di log non sia stato specificato alcun nome logico. Se non viene specificato un nome logico di file di log, ne viene generato uno automaticamente tramite l'aggiunta di un suffisso all'argomento *database_name*. Questo limita il numero di caratteri dell'argomento *database_name* a 123 per fare in modo che il nome logico di file di log generato includa meno di 128 caratteri.
ON
Specifica che i file su disco utilizzati per archiviare le sezioni di dati del database (file di dati) vengono definiti in modo esplicito. Alla parola chiave segue un elenco delimitato da virgola di voci <filespec> che definiscono i file di dati del filegroup primario. L'elenco di file del filegroup primario può essere seguito da un elenco facoltativo delimitato da virgola di voci <filegroup> che definiscono i filegroup utente e i relativi file.
n
Segnaposto che indica la possibilità di specificare più file per il nuovo database.
LOG ON
Specifica che i file su disco utilizzati per archiviare il log del database (file di log) vengono definiti in modo esplicito. Alla parola chiave segue un elenco delimitato da virgola di voci <filespec> che definiscono i file di log. Se la parola chiave LOG ON viene omessa, viene creato automaticamente un singolo file di log generato dal sistema con dimensioni pari al 25% della somma delle dimensioni di tutti i file di dati del database.
FOR LOAD
Questa clausola è supportata per compatibilità con versioni precedenti di Microsoft SQL Server. Il database viene creato con l'opzione di database **dbo use only** attivata e lo stato viene impostato per il caricamento. Ciò non è necessario in SQL Server versione 7.0, in cui un database può essere ricreato tramite l'istruzione RESTORE come parte dell'operazione di ripristino.
FOR ATTACH
Specifica che un database viene collegato da un set di file del sistema operativo già esistente. È necessario che una voce dell'elenco <filespec> specifichi il primo file primario. Le altre voci <filespec> necessarie sono quelle relative ai file con percorso diverso rispetto al percorso utilizzato in fase di creazione del database o quando il database è stato scollegato. Per questi file è necessario specificare una voce <filespec>. Il database collegato deve essere stato creato in base alla stessa tabella codici e allo stesso tipo di ordinamento di SQL Server. Utilizzare la stored procedure di sistema

sp_attach_db anziché eseguire direttamente l'istruzione CREATE DATABASE FOR ATTACH.
Utilizzare l'istruzione CREATE DATABASE FOR ATTACH solo quando è necessario specificare più di 16 voci <filespec>.
Se si collega un database a un server diverso dal server da cui il database è stato scollegato e il database scollegato è abilitato per la replica, è necessario eseguire **sp_removedbreplication** per rimuovere le opzioni di replica dal database.
collation_name
Specifica le regole di confronto predefinite per il database. È possibile utilizzare nomi di regole di confronto di Windows o SQL. Se *collation_name* viene omesso, al database vengono assegnate le regole di confronto predefinite dell'istanza di SQL Server.

PRIMARY
Specifica che l'elenco <filespec> associato definisce il file primario. Il filegroup primario include tutte le tabelle di sistema del database e tutti gli oggetti non assegnati ai filegroup utente. La prima voce di <filespec> del filegroup primario diventa il file primario, ovvero il file contenente l'inizio logico del database e le tabelle di sistema. Un database può includere un solo file primario. Se la parola chiave PRIMARY viene omessa, il primo file elencato nell'istruzione CREATE DATABASE diventa il file primario.
NAME
Specifica il nome logico del file definito in <filespec>. Il parametro NAME non è necessario quando viene specificata la clausola FOR ATTACH.
logical_file_name
Nome utilizzato per fare riferimento al file in qualsiasi istruzione Transact-SQL eseguita dopo la creazione del database. *logical_file_name* deve essere univoco nel database e conforme alle regole per gli identificatori. Il nome può essere un carattere o una costante Unicode oppure un identificatore normale o delimitato.
FILENAME
Specifica il nome di file del sistema operativo definito in <filespec>.
'*os_file_name***'**
Percorso e nome di file utilizzati dal sistema operativo in fase di creazione del file fisico definito in <filespec>. Il percorso specificato in *os_file_name* deve corrispondere a una directory di un'istanza di SQL Server. Non è consentito specificare una directory di un file system compresso.
Se il file viene creato in una partizione non formattata, *os_file_name* deve specificare solo la lettera dell'unità di una partizione non formattata esistente. È possibile creare un solo file in ogni partizione non formattata. Le dimensioni dei file di partizioni non formattate non aumentano automaticamente. I parametri MAXSIZE e FILEGROWTH pertanto non sono necessari quando *os_file_name* specifica una partizione non formattata.
SIZE
Specifica le dimensioni del file definito in <filespec>. Se il parametro SIZE viene omesso dalla voce <filespec> per un file primario, vengono utilizzate le dimensioni del file primario del database **model**. Se SIZE viene omesso dalla voce <filespec> per un file secondario, viene creato un file di 1 MB.
size
Dimensioni iniziali del file definito in <filespec>. È possibile utilizzare i suffissi per kilobyte (KB), megabyte (MB), gigabyte (GB) e terabyte (TB). Il valore predefinito è MB. Specificare un numero intero, ovvero non includere decimali. Il valore minimo consentito per l'argomento *size* è 512 KB. Se *size* viene omesso, il valore predefinito è 1 MB. Le dimensioni specificate per il file di dati primario deve essere uguale almeno alle dimensioni del file primario del database **model**.
MAXSIZE
Specifica le dimensioni massime per il file definito in <filespec>.
max_size
Dimensioni massime per il file definito in <filespec>. È possibile utilizzare i suffissi per kilobyte (KB), megabyte (MB), gigabyte (GB) e terabyte (TB). Il valore predefinito è MB. Specificare un numero intero, ovvero non includere decimali. Se *max_size* viene omesso, le dimensioni del file aumentano fino a quando il disco non risulta pieno.
Nota Quando un disco è quasi pieno, l'amministratore di sistema di SQL Server riceve un avviso inviato dal registro di sistema S/B di Microsoft Windows NT®.
UNLIMITED
Specifica che le dimensioni del file definito in <filespec> aumentano fino a quando il disco risulta pieno.
FILEGROWTH
Specifica l'incremento delle dimensione del file definito in <filespec>. Il valore impostato per il parametro FILEGROWTH di un file non può essere superiore al valore del parametro MAXSIZE.

growth_increment
Quantità di spazio aggiunta al file ogni volta che è necessario spazio aggiuntivo. Specificare un numero intero, ovvero non includere decimali. Il valore 0 indica che le dimensioni non verranno aumentate. È possibile specificare il valore in megabyte (MB), kilobyte (KB), gigabyte (GB) o terabyte (TB) oppure in forma di percentuale (%). Se si specifica un valore senza il suffisso MB, KB o %, il suffisso predefinito è MB. Se si utilizza il suffisso %, l'incremento corrisponde alla percentuale delle dimensioni del file specificata quando si verifica l'incremento. Se FILEGROWTH viene omesso, il valore predefinito è 10% e il valore minimo è 64 KB. Le dimensioni specificate vengono arrotondate al blocco di 64 KB più prossimo.

Osservazioni

È possibile utilizzare un'istruzione CREATE DATABASE per creare un database e i file in cui archiviarlo. In SQL Server l'istruzione CREATE DATABASE è implementata in due passaggi:

1. Viene innanzitutto utilizzata una copia del database **model** per inizializzare il database e i metadati corrispondenti.

2. La parte rimanente del database viene quindi compilata con pagine vuote, ad eccezione delle pagine con dati interni che registrano la modalità di utilizzo dello spazio nel database.

Gli oggetti definiti dall'utente inclusi nel database **model** vengono pertanto copiati in tutti i nuovi database. È possibile aggiungere al database **model** qualsiasi oggetto che si desidera includere in tutti i database, ad esempio tabelle, viste, stored procedure, tipi di dati e così via.
Ogni nuovo database eredita le impostazioni delle opzioni di database del database **model** (a meno che non venga specificata l'opzione FOR ATTACH). L'opzione di database **select into/bulkcopy**, ad esempio, è impostata su OFF nel database **model** e in tutti i nuovi database. Se si esegue l'istruzione ALTER DATABASE per modificare le opzioni del database **model**, tali impostazioni vengono adottate anche nei nuovi database. Se viene specificata l'opzione FOR ATTACH nell'istruzione CREATE DATABASE, i nuovi database ereditano le impostazioni delle opzioni di database dal database originale.
In un server è possibile specificare al massimo 32.767 database.
Per l'archiviazione di un database vengono utilizzati tre tipi di file:

- Il file primario include le informazioni di avvio del database. Viene inoltre utilizzato per archiviare dati. A ogni database è associato un file primario.

- I file secondari contengono tutti i dati che non è possibile includere nel file primario. I file secondari non sono necessari se il file di dati primario è sufficientemente grande per contenere tutti i dati del database. Per alcuni database di grandi dimensioni potrebbe essere necessario utilizzare più file secondari oppure file secondari su unità disco distinte in modo da distribuire i dati su più dischi.

- I file di log delle transazioni contengono le informazioni necessarie per il ripristino del database. È necessario che per ogni database sia disponibile almeno un file di log delle transazioni. La dimensione minima per un file di log delle transazioni è 512 KB.

Ogni database include almeno due file, un file primario e un file di log delle transazioni.
Sebbene **'***os_file_name***'** possa essere qualsiasi nome valido di file del sistema operativo, è consigliabile specificare le seguenti estensioni in modo che il nome indichi chiaramente lo scopo del file.

Tipo di file	Estensione del nome del file
File di dati primario	mdf
File di dati secondario	ndf
File di log delle transazioni	ldf

Quando una semplice istruzione CREATE DATABASE *database_name* viene specificata senza parametri aggiuntivi, il database viene creato con le stesse dimensioni del database **model**.
A tutti i database è associato almeno un filegroup primario, a cui vengono assegnate tutte le tabelle di sistema. Per un database potrebbero essere inoltre disponibili filegroup definiti dall'utente. Se un oggetto viene creato con una clausola ON *filegroup* che specifica un filegroup definito dall'utente, tutte le pagine per l'oggetto vengono allocate nel filegroup specificato. Le pagine per tutti gli oggetti utente creati senza la clausola ON *filegroup* o con la clausola ON DEFAULT vengono allocate nel filegroup

primario. Nei nuovi database il filegroup primario è il filegroup predefinito. È tuttavia possibile impostare come predefinito un filegroup definito dall'utente tramite l'istruzione ALTER DATABASE:
ALTER DATABASE *database_name* MODIFY FILEGROUP *filegroup_name* DEFAULT
Ogni database ha un proprietario che può eseguire attività particolari nel database. Il proprietario è l'utente che crea il database. È possibile modificare il proprietario del database con la procedura **sp_changedbowner**.
Per visualizzare un report relativo a un database o a tutti i database di un computer SQL Server, utilizzare la procedura **sp_helpdb**. Per ottenere un report relativo allo spazio in uso in un database, utilizzare la procedura **sp_spaceused**. Per un report relativo ai filegroup di un database, utilizzare la procedura **sp_helpfilegroup**, mentre per un report dei file di un database utilizzare la procedura **sp_helpfile**.
Nelle versioni precedenti di SQL Server, prima di eseguire l'istruzione CREATE DATABASE vengono eseguite istruzioni DISK INIT per creare i file di un database. Per compatibilità con le versioni precedenti, l'istruzione CREATE DATABASE consente di creare un nuovo database in file o dispositivi creati con l'istruzione DISK INIT.

Autorizzazioni

L'autorizzazione per l'istruzione CREATE DATABASE viene assegnata per impostazione predefinita ai membri dei ruoli predefiniti del server **sysadmin** e **dbcreator**. I membri dei ruoli predefiniti del server **sysadmin** e **securityadmin** possono concedere autorizzazioni per l'istruzione CREATE DATABASE ad altri account di accesso. I membri dei ruoli predefiniti del server **sysadmin** e **dbcreator** possono aggiungere altri account di accesso al ruolo **dbcreator**. È necessario che l'autorizzazione per l'istruzione CREATE DATABASE sia concessa in modo esplicito. Non è possibile concederla tramite l'istruzione GRANT ALL.
L'autorizzazione per l'istruzione CREATE DATABASE viene normalmente limitata a pochi account di accesso per mantenere il controllo sull'utilizzo del disco di un computer SQL Server.

Esempi

A. Creazione di un database che specifica i file di dati e i file di log delle transazioni

In questo esempio viene creato il database **Sales**. Dato che la parola chiave PRIMARY non è specificata, il primo file, ovvero **Sales_dat**, corrisponde al file primario. Nel parametro SIZE non viene specificato il suffisso MB o KB per le dimensioni del file **Sales_dat**, che per impostazione predefinita vengono pertanto allocate in megabyte. Le dimensioni del file **Sales_log** vengono allocate in megabyte perché nel parametro SIZE è stato specificato in modo esplicito il suffisso MB.

```
USE master
GO
CREATE DATABASE Sales
ON
( NAME = Sales_dat,
   FILENAME = 'c:\program files\microsoft sql server\mssql\data\saledat.mdf',
   SIZE = 10,
   MAXSIZE = 50,
   FILEGROWTH = 5 )
LOG ON
( NAME = 'Sales_log',
   FILENAME = 'c:\program files\microsoft sql server\mssql\data\salelog.ldf',
   SIZE = 5MB,
   MAXSIZE = 25MB,
   FILEGROWTH = 5MB )
GO
```

B. Creazione di un database specificando più file di dati e più file di log delle transazioni

In questo esempio viene creato il database **Archive** con tre file di dati da 100 MB e due file di log delle transazioni da 100 MB. Il file primario è il primo file dell'elenco e viene specificato in modo esplicito con la parola chiave PRIMARY. I file di log delle transazioni vengono specificati dopo le parole chiave LOG ON. Si notino le estensioni utilizzate per i file nell'opzione FILENAME: mdf per i file di dati primari, ndf per i file di dati secondari e ldf per i file di log delle transazioni.

```
USE master
GO
CREATE DATABASE Archive
```

```
ON
PRIMARY ( NAME = Arch1,
     FILENAME = 'c:\program files\microsoft sql server\mssql\data\archdat1.mdf',
     SIZE = 100MB,
     MAXSIZE = 200,
     FILEGROWTH = 20),
( NAME = Arch2,
  FILENAME = 'c:\program files\microsoft sql server\mssql\data\archdat2.ndf',
  SIZE = 100MB,
  MAXSIZE = 200,
  FILEGROWTH = 20),
( NAME = Arch3,
  FILENAME = 'c:\program files\microsoft sql server\mssql\data\archdat3.ndf',
  SIZE = 100MB,
  MAXSIZE = 200,
  FILEGROWTH = 20)
LOG ON
( NAME = Archlog1,
  FILENAME = 'c:\program files\microsoft sql server\mssql\data\archlog1.ldf',
  SIZE = 100MB,
  MAXSIZE = 200,
  FILEGROWTH = 20),
( NAME = Archlog2,
  FILENAME = 'c:\program files\microsoft sql server\mssql\data\archlog2.ldf',
  SIZE = 100MB,
  MAXSIZE = 200,
  FILEGROWTH = 20)
GO
```

C. Creazione di un database semplice

In questo esempio viene creato il database **Products** e viene specificato un singolo file. Il file specificato diventa il file primario e viene automaticamente creato un file di log delle transazioni da 1 MB. Poiché per il file primario non viene specificato il suffisso MB o KB nel parametro **SIZE**, le dimensioni del file vengono allocate in megabyte. Inoltre, dato che per il file di log delle transazioni non è specificata alcuna voce <filespec>, il parametro MAXSIZE non viene specificato, ovvero le dimensioni del file di log delle transazioni potranno aumentare fino a riempire lo spazio disponibile del disco.

```
USE master
GO
CREATE DATABASE Products
ON
( NAME = prods_dat,
  FILENAME = 'c:\program files\microsoft sql server\mssql\data\prods.mdf',
  SIZE = 4,
  MAXSIZE = 10,
  FILEGROWTH = 1 )
GO
```

D. Creazione di un database senza specificare alcun file

In questo esempio viene creato il database **mytest** insieme al file primario e al file di log delle transazioni corrispondente. Poiché nell'istruzione non è specificata alcuna voce <filespec>, le dimensioni del file primario del database corrispondono a quelle del file primario del database **model**, così come le dimensioni del file di log delle transazioni, che corrispondono a quelle del file di log delle transazioni del database **model**. Poiché MAXSIZE non è specificato, le dimensioni dei file possono aumentare fino a riempire lo spazio disponibile su disco.

```
CREATE DATABASE mytest
```

E. Creazione di un database senza specificare il parametro SIZE

In questo esempio viene creato il database **products2**. Il file **prods2_dat** diventa il file primario con dimensioni pari a quelle del file primario del database **model**. Il file di log delle transazioni viene creato automaticamente con dimensioni pari al 25% delle dimensioni del file primario o a 512 KB, a seconda

del valore maggiore. Poiché MAXSIZE non è specificato, le dimensioni dei file possono aumentare fino a riempire lo spazio disponibile su disco.

```
USE master
GO
CREATE DATABASE Products2
ON
( NAME = prods2_dat,
  FILENAME = 'c:\program files\microsoft sql server\mssql\data\prods2.mdf' )
GO
```

F. Creazione di un database con filegroup

In questo esempio viene creato il database **sales** con i tre seguenti filegroup:

- Il filegroup primario con i file **Spri1_dat** e **Spri2_dat**. L'incremento specificato nel parametro FILEGROWTH per tali file è uguale al 15%.
- Il filegroup **SalesGroup1** con i file **SGrp1Fi1** e **SGrp1Fi2**.
- Il filegroup **SalesGroup2** con i file **SGrp2Fi1** e **SGrp2Fi2**.

```
CREATE DATABASE Sales
ON PRIMARY
( NAME = SPri1_dat,
  FILENAME = 'c:\program files\microsoft sql server\mssql\data\SPri1dat.mdf',
  SIZE = 10,
  MAXSIZE = 50,
  FILEGROWTH = 15% ),
( NAME = SPri2_dat,
  FILENAME = 'c:\program files\microsoft sql server\mssql\data\SPri2dt.ndf',
  SIZE = 10,
  MAXSIZE = 50,
  FILEGROWTH = 15% ),
FILEGROUP SalesGroup1
( NAME = SGrp1Fi1_dat,
  FILENAME = 'c:\program files\microsoft sql server\mssql\data\SG1Fi1dt.ndf',
  SIZE = 10,
  MAXSIZE = 50,
  FILEGROWTH = 5 ),
( NAME = SGrp1Fi2_dat,
  FILENAME = 'c:\program files\microsoft sql server\mssql\data\SG1Fi2dt.ndf',
  SIZE = 10,
  MAXSIZE = 50,
  FILEGROWTH = 5 ),
FILEGROUP SalesGroup2
( NAME = SGrp2Fi1_dat,
  FILENAME = 'c:\program files\microsoft sql server\mssql\data\SG2Fi1dt.ndf',
  SIZE = 10,
  MAXSIZE = 50,
  FILEGROWTH = 5 ),
( NAME = SGrp2Fi2_dat,
  FILENAME = 'c:\program files\microsoft sql server\mssql\data\SG2Fi2dt.ndf',
  SIZE = 10,
  MAXSIZE = 50,
  FILEGROWTH = 5 )
LOG ON
( NAME = 'Sales_log',
  FILENAME = 'c:\program files\microsoft sql server\mssql\data\salelog.ldf',
  SIZE = 5MB,
  MAXSIZE = 25MB,
  FILEGROWTH = 5MB )
GO
```

G. Collegamento di un database

Nell'esempio B viene creato il database **Archive** con i seguenti file fisici:

c:\program files\microsoft sql server\mssql\data\archdat1.mdf
c:\program files\microsoft sql server\mssql\data\archdat2.ndf
c:\program files\microsoft sql server\mssql\data\archdat3.ndf
c:\program files\microsoft sql server\mssql\data\archlog1.ldf
c:\program files\microsoft sql server\mssql\data\archlog2.ldf
È possibile scollegare il database con la stored procedure **sp_detach_db**, quindi ricollegarlo utilizzando l'istruzione CREATE DATABASE con la clausola FOR ATTACH:

```
sp_detach_db Archive
GO
CREATE DATABASE Archive
ON PRIMARY (FILENAME = 'c:\program files\microsoft sql server\mssql\data\archdat1.mdf')
FOR ATTACH
GO
```

H. Utilizzo di partizioni non formattate

In questo esempio viene creato il database **Employees** utilizzando partizioni non formattate. È necessario che le partizioni non formattate esistano quando si esegue l'istruzione. Ogni partizione non formattata può includere un solo file.

```
USE master
GO
CREATE DATABASE Employees
ON
( NAME = Empl_dat,
   FILENAME = 'f:',
   SIZE = 10,
   MAXSIZE = 50,
   FILEGROWTH = 5 )
LOG ON
( NAME = 'Sales_log',
   FILENAME = 'g:',
   SIZE = 5MB,
   MAXSIZE = 25MB,
   FILEGROWTH = 5MB )
GO
```

I. Utilizzo di unità installate

In questo esempio viene creato il database **Employees** utilizzando unità installate che puntano a partizioni non formattate. Questa funzione è disponibile solo in Microsoft® Windows®. È necessario che, quando si esegue l'istruzione, le unità installate e le partizioni non formattate siano esistenti. Inoltre, ogni partizione è possibile inserire un solo file. Quando si crea un file di database in un'unità installata, il percorso dell'unità deve terminare con una barra rovesciata (\).

```
USE master
GO
CREATE DATABASE Employees
ON
( NAME = Empl_dat,
   FILENAME = 'd:\sample data dir\',
   SIZE = 10,
   MAXSIZE = 50,
   FILEGROWTH = 5 )
LOG ON
( NAME = 'Sales_log',
   FILENAME = 'd:\sample log dir\',
   SIZE = 5MB,
   MAXSIZE = 25MB,
   FILEGROWTH = 5MB )
GO
```

Creazioni tabelle

Per creare nuove tabelle in un database, è necessario utilizzare l'istruzione CREATE TABLE. L'istruzione completa accetta argomenti per il nome della tabella. Per ciascuna colonna (campo) è necessario aggiungere una serie di argomenti relativi al nome del campo, al tipo di dati e, per le colonne di testo, alla dimensione espressa in caratteri.

```
CREATE TABLE [[database.]owner.]table_name
(column_name datatype [not null | null] IDENTITY[(seed,
increment)][constraint]
[, column_name datatype [not null | null IDENTITY[(seed, increment)]]].
[constraint]...)
 [ON segment name]
```

Nell'esempio che segue viene creata la tabella Impiegati con due colonne di testo contenenti 25 caratteri ciascuna:

```
CREATE TABLE Utente.Impiegati (
NOME CHAR (25),
COGNOME CHAR (25));

The command(s) completed successfully.
```

Attenzione: la tabella creata ha un proprietario! (nel nostro caso il generico **utente**)

Eliminare una tabella

Drop Table Utente.Impiegati;

The command(s) completed successfully.

Aggiunta e rimozione di colonne

Per aggiungere, modificare o rimuovere colonne da una tabella, è necessario utilizzare l'istruzione ALTER TABLE. La seguente istruzione, ad esempio, consente di aggiungere alla tabella Impiegati una colonna di testo contenente 25 caratteri denominata Note:

```
ALTER TABLE utente.Impiegati ADD Note VARCHAR (200);

The command(s) completed successfully.
```

Per modificare una colonna utilizzare modify.

```
ALTER TABLE utente.Impiegati ADD indirizzo varchar (20);

ALTER TABLE utente.Impiegati alter column indirizzo varchar (40);
```

Per eliminare colonna:
ALTER TABLE utente.Impiegati drop column indirizzo;

Creazione ed eliminazione di indici

Per creare un indice, è possibile procedere in uno dei seguenti modi:

- Utilizzare l'istruzione CREATE TABLE al momento della creazione della tabella.
- Utilizzare l'istruzione CREATE INDEX.

Pur producendo risultati analoghi, i 2 metodi presentano alcune differenze. Talvolta può essere preferibile creare una tabella che inizialmente non contenga indici e quindi progettare i parametri dell'indice dopo aver utilizzato il modello della tabella. In questo caso è necessario utilizzare l'istruzione CREATE TABLE per creare il modello della tabella senza indici e quindi aggiungervi gli indici desiderati utilizzando un'istruzione CREATE INDEX o ALTER TABLE.

Gli indici in SQLServer possono essere *Clustered* o *non Clustered*.

Un indice *Clustered* (raggruppato) usa un algortmo di ricerca basato su un albero binario e implica che SQLServer ordinerà fisicamente i recrd in base all'ordine dei valori presenti in una determinata colonna o in più colonne di datiSi può avere un solo indice raggruppato per tabella.Un indice raggruppato viene di solito associato alla chiave primaria della tabella.

Un indice non *Clustered* (non raggruppato) usa anch'esso un algoritmo di ricerca basato su un albero binario, ma non ordina fisicamente i record.Di conseguenza l'attività di interrogazione è solitamente più veloce ma meno di un indice Clustered.

Creazione di un indice con l'istruzione CREATE TABLE

Quando si crea una tabella è possibile creare un indice per ciascuna colonna oppure un solo indice per più colonne utilizzando la proposizione SQL CONSTRAINT. La parola chiave CONSTRAINT compare all'inizio della definizione di un indice. Nell'esempio che segue vengono illustrate le modalità di creazione di una tabella con un indice su tre colonne:

```
CREATE TABLE UTENTE.Impiegati (
Nome CHAR (25) not null,
Cognome CHAR (25) not null,
Soprannome varchar (80) not null,
Data_di_nascita DATETIME,
indirizzo Varchar (200),
CAP Char (5),
Comune varchar (50),
Provincia char (2),
Unique (Nome,Cognome,Soprannome),
CONSTRAINT IndiceImpiegati Primary key CLUSTERED (Nome, Cognome,
Data_di_nascita));
```

Per indicizzare una sola colonna, è necessario inserire la proposizione CONSTRAINT in una sola definizione di colonna. Per indicizzare, ad esempio, solo la colonna Data di nascita, è necessario utilizzare la seguente istruzione CREATE TABLE:

```
CREATE TABLE UTENTE.Impiegati (Nome CHAR (25),
Cognome CHAR (25) Not null,
Data_di_nascita DATETIME,
IDImpiegati integer CONSTRAINT Primario PRIMARY key);
```

```
CREATE TABLE UTENTE.Impiegato  (Nome CHAR (25),
Cognome CHAR (25) Not null,
Data_di_nascita DATETIME,
IDImpiegati integer PRIMARY key);

CREATE TABLE UTENTE.Impiegato  (Nome CHAR (25),
Cognome CHAR (25) Not null,
Data_di_nascita DATETIME,
IDImpiegati integer PRIMARY key CLUSTERED);
```

A differenza degli indici a colonne multiple, per gli indici a colonna singola la parola chiave CONSTRAINT, che compare all'inizio della definizione dell'indice, non è separata dall'ultima colonna per mezzo di una virgola, ma segue immediatamente il tipo di dati della colonna indicizzata.

Nota. La chiave è un campo non una combinazione di campi che devono avere questi attributi:

devono essere univoci.

Non devono essere nulli.

Creazione di un indice con l'istruzione CREATE INDEX

Per creare un indice è inoltre possibile utilizzare l'istruzione CREATE INDEX. L'esempio che segue produce lo stesso risultato dell'esempio precedente, con la differenza che in questo caso viene utilizzata l'istruzione CREATE INDEX fuori dalla creazione di una tabella:

```
CREATE UNIQUE INDEX MyIndex ON utente.Impiegato (Data_di_nascita);

CREATE NonClustered INDEX MyIndex ON utente.Impiegato (Data_di_nascita);
```

Nota per usare il comando create index bisogna essere proprietari della tabella.

Se una tabella subisce molti aggiornamenti gli indici possono perdere di efficacia.

Per eliminare un indice il codice è:

DROP INDEX Proprietario.Tabella.MyIndex;

DROP INDEX utente.Impiegato.myindex;

Chiave Primaria ed integrità referenziale

La chiave primaria è quell'attributo o quella combinazione di attributi che individuano univocamente un record.

```
CREATE TABLE STRUTTURA
      (
      CODSTRUTTURA VARCHAR(10),
      DENOMINAZIONE VARCHAR(150),
      INDIRIZZO VARCHAR(150),
```

```
      CAP VARCHAR (5),
      COMUNE VARCHAR (30),
      PROVINCIA VARCHAR (2),
      FAX VARCHAR(14),
      TEL VARCHAR(14),
      ANNOTA VARCHAR (255),
      PRIMARY KEY (CODSTRUTTURA)
      );


CREATE TABLE STRUTTURA (
      CODSTRUTTURA VARCHAR(10),
      DENOMINAZIONE VARCHAR(150),
      INDIRIZZO VARCHAR(150),
      CAP VARCHAR (5),
      COMUNE VARCHAR (30),
      PROVINCIA VARCHAR (2),
      FAX VARCHAR(14),
      TEL VARCHAR(14),
      ANNOTA VARCHAR (255),
      CONSTRAINT Primarykey PRIMARY KEY (codstruttura)
      );



 CREATE TABLE STRUTTURA (
      CODSTRUTTURA VARCHAR(10) PRIMARY KEY CLUSTERED,
      DENOMINAZIONE VARCHAR(150),
      INDIRIZZO VARCHAR(150),
      CAP VARCHAR (5),
      COMUNE VARCHAR (30),
      PROVINCIA VARCHAR (2),
      FAX VARCHAR(14),
      TEL VARCHAR(14),
      ANNOTA VARCHAR (255)
      );
```

La chiave esterna è una combinazione di colonne con valori basati su quelli della chiave primaria di un'altra tabella.

CONSTRAINT constraint_name FOREIGN KEY (column_name_1 column_name_n) REFERENCES _table_name (column_name_1 column_name_n)

```
 CREATE TABLE LAVORATORE(
      CODLAV VARCHAR(10),
      CODSTRUTTURA VARCHAR(10),
      NOME VARCHAR(20),
      COGNOME VARCHAR(20),
      CODQUAL VARCHAR(10),
      CODSEDEPOSTA VARCHAR(10),
      DESCRSEDEPOSTA VARCHAR(100),
      INDIRIZZO VARCHAR(150),
      COMUNE VARCHAR(30),
      CAP VARCHAR(5),
```

```
PROVINCIA VARCHAR (2),
FAX VARCHAR(14),
TEL VARCHAR(14),
POSTA VARCHAR(100),
PRIMARY KEY (CODLAV),
FOREIGN KEY (CODSTRUTTURA) REFERENCES STRUTTURA
);
```

7. Operazioni DML

Le istruzioni del linguaggio di gestione dei dati SQL, dette anche DML, consentono di aggiornare i record e di recuperare, aggiungere o eliminare record da una tabella. Per tali operazioni sono supportate numerose istruzioni, la maggior parte delle quali rientra nella struttura generale della query SELECT.

La query SELECT

L'istruzione SELECT consente di recuperare un gruppo di record da un database e di memorizzarlo in un nuovo oggetto che costituisce un nuovo set di dati. Un'applicazione può gestire tale oggetto Recordset visualizzando, aggiungendo, modificando e/o eliminando i record necessari, nonché produrre e visualizzare report in base ai dati.

SELECT è in genere la prima parola di un'istruzione SQL. La maggior parte delle istruzioni SQL sono istruzioni SELECT. Le istruzioni SELECT consentono di recuperare i dati contenuti in un database, senza tuttavia modificarli.

Di seguito è riportata la sintassi generale della query SELECT:

SELECT *elencocampi*
FROM *nomitabelle*
WHERE *condizioniricerca*
GROUP BY *elencocampi*
HAVING *criteridiraggruppamento*
ORDER BY *elencocampi*

Le istruzioni e proposizioni verranno descritte nelle sezioni successive.

La query di base

La query SELECT più semplice è la seguente:

SELECT * FROM *nometabella*

La query SELECT riportata di seguito, ad esempio, restituisce tutte le colonne di tutti i record della tabella Impiegati:

```
SELECT *
FROM DBO.employees;
```

L'asterisco indica che verranno recuperate tutte le colonne della tabella desiderata. Se lo si desidera è possibile specificare solo alcune colonne. I dati di ciascuna colonna verranno visualizzati nell'ordine in cui sono elencati. In questo modo è possibile riordinare le colonne per rendere la tabella più leggibile:

```
SELECT Lastname,firstname
```

```
FROM dbo.employees;
```

```
Lastname             firstname
-------------------- ----------
Davolio              Nancy
Fuller               Andrew
Leverling            Janet
Peacock              Margaret
Buchanan             Steven
Suyama               Michael
King                 Robert
Callahan             Laura
Dodsworth            Anne

(9 row(s) affected)
```

Indicazione dell'origine dei dati

Un'istruzione SELECT deve contenere una proposizione FROM che specifica la tabella da cui verranno prelevati i record.

Se il nome di un campo è incluso in più tabelle specificate dalla proposizione FROM, dovrà essere preceduto dal nome della tabella e dall'operatore "." (punto). Nell'esempio che segue, il campo `job_id` è contenuto sia nella tabella `employee` che nella tabella `job`.

```
select employees.Lastname,Employees.FirstName,Orders.Orderdate
from Employees,Orders
where Employees.Employeeid=Orders.Employeeid;
```

Se la proposizione FROM specifica più tabelle, l'ordine in cui esse appaiono non è rilevante.

Applicazione di filtri ed ordinamento dei risultati della query

Nelle sezioni successive vengono descritte le proposizioni e i predicati di parole chiave facoltative del linguaggio SQL che consentono di definire ulteriormente una query e di ordinarne il risultato.

Predicato DISTINCT

Per evitare che nella stessa colonna compaiano record contenenti dati duplicati è possibile utilizzare la parola chiave DISTINCT. Per poter essere inclusi nei risultati della query, i valori di ciascuna colonna o combinazione di colonne elencati nell'istruzione SELECT devono essere univoci. È possibile, ad esempio, che nella tabella Impiegati siano elencate più persone con lo stesso cognome. Se si prende la select precedente e non si visualizza la data dell'ordine otterrò un alenco lungo in cui i nomi degli impiegati si ripetono.Il distinct compatta questa selezione riportando solo re righe differenti.

```
select distinct employees.Lastname,Employees.FirstName
from Employees,Orders
```

```
where Employees.Employeeid=Orders.Employeeid;

Lastname             FirstName
-------------------- ----------
Fuller               Andrew
Dodsworth            Anne
Leverling            Janet
Callahan             Laura
Peacock              Margaret
Suyama               Michael
Davolio              Nancy
King                 Robert
Buchanan             Steven

(9 row(s) affected)
```

Se la parola chiave DISTINCT viene omessa, la query restituirà solo i record contenenti nome e cognome diversi.

Proposizione WHERE

La proposizione WHERE indica quali tra i record della tabella specificata dalla proposizione FROM verranno inclusi nei risultati dell'istruzione SELECT.

Il modulo di gestione di database seleziona i record che soddisfano le condizioni elencate nella proposizione WHERE. Se quest'ultima viene omessa, la query restituirà tutte le righe della tabella. Se nella query vengono specificate più tabelle senza includere una proposizione WHERE o JOIN, la query genererà il prodotto cartesiano delle tabelle. Le condizioni della clausola WHERE sono specificate per mezzo degli operatori di confronto semplici, dei connettori logici e degli operatori BETWEEN, IN, LIKE, IS NULL.

Nota Benché la proposizione WHERE possa eseguire operazioni analoghe, se si desidera che il Recordset risultante sia aggiornabile è necessario utilizzare la proposizione JOIN per eseguire operazioni di join SQL su tabelle multiple.

La proposizione WHERE è facoltativa. Quando viene inclusa deve seguire la proposizione FROM. È possibile, ad esempio, selezionare tutti gli impiegati del reparto vendite:

```
WHERE Rep = 'Vendite'
```

oppure tutti i clienti di età compresa tra 18 e 30 anni:

```
WHERE Anni BETWEEN 18 AND 30
```

Le proposizioni WHERE e HAVING sono simili. La prima consente di determinare quali record verranno selezionati e la seconda determina quali dei record raggruppati con la proposizione GROUP BY verranno visualizzati.

La proposizione WHERE consente di eliminare i record che non si desidera vengano raggruppati dalla proposizione GROUP BY.

È possibile utilizzare diverse espressioni per determinare quali record verranno restituiti dall'istruzione SQL. La seguente istruzione SQL, ad esempio, seleziona tutti gli impiegati con retribuzione superiore a $. 1.500:

```
SELECT Categories.CategoryName, Categories.Description,
    Products.ProductName
FROM Categories INNER JOIN
    Products ON
    Categories.CategoryID = Products.CategoryID
```

```
WHERE (Categories.CategoryName = N'beverages')
```

Gli operatori di confronto

Impostare un confronto vuol dire definire le condizioni che devono essere utilizzate per individuare le righe volute. Possiamo quindi affermare che un confronto è una forma sintattica che può assumere i classici valori booleani (vero o falso)[14]*.* Il risultato del confronto seguirà quindi le regole delle tavole della verità di Boole

=, >,<,<> ,>= ,<= I due elementi posti a confronto devono essere compatibili, ovvero numerici o alfanumerici. Con questa operazione si effettua una selezione di righe.

```
SELECT *
FROM p
WHERE peso >6000
```

CodP	NomeP	Colore	Peso	Città
P2	bullone	verde	7718	Parigi
P3	vite	blu	7718	Roma
P4	vite	rosso	6356	Londra
P6	dente	rosso	8626	Londra

Connettori logici

Per mezzo dei connettori logici è possibile realizzare delle operazioni di confronto complesse combinando più condizioni insieme. Gli operatori in questione sono AND, OR, NOT che vengono usati per combinare fra loro confronti semplici usando eventualmente delle parentesi per stabilire l'ordine di valutazione. L'ordine di valutazione degli operatori prevede che vengano prima applicati gli operatori NOT, poi gli AND ed infine gli OR. Quindi, per definire un ordine di valutazione diverso occorre usare le parentesi tonde.

AND L'operatore AND restituisce vero quando entrambe le condizioni che lega sono vere (P1*P2)

```
SELECT *
FROM f
WHERE livello >10 AND città = 'Londra'
```

CodF	NomeF	Livello	Città
F1	Sala	20	Londra
F4	Carli	20	Londra

OR L'operatore OR restituisce vero quando almeno una delle condizioni, quella che precede o quella che segue sono vere (P1+P2)

```
SELECT *
FROM f
WHERE livello >10 OR città = 'Londra'
```

CodF	NomeF	Livello	Città
F1	Sala	20	Londra
F3	Bocca	30	Parigi
F4	Carli	20	Londra
F5	Amici	30	Atene

NOT L'operatore NOT restituisce vero quando la condizione che segui è falsa

```
SELECT *
FROM f
WHERE NOT (livello >10 AND città = 'Londra')
```

CodF	NomeF	Livello	Città
F2	Goggi	10	Parigi
F3	Bocca	30	Parigi
F5	Amici	30	Atene

Gli operatori BETWEEN, IN, LIKE

BETWEEN L'operatore BETWEEN verifica se un argomento è compreso in un intervallo di valori. La sintassi è la seguente :
operando BETWEEN operando AND operando.
L'operatore BETWEEN riesce a realizzare una condizione più sintetica di quella che si otterrebbe con il solo AND. Spesso però viene preferita la seconda soluzione quando la complessità della interrogazione tende ad aumentare poiché l'uso di BETWEEN prevede che gli

```
SELECT Nomep, peso, città
FROM P
WHERE peso BETWEEN 4000 AND 5500
```

Utilizzando solo AND si avrebbe avuto :

```
SELECT Nomep, peso, città
FROM P
WHERE peso>= 4000 AND peso<= 5500
```

NomeP	Peso	Città
Dado	5448	Londra
camma	5448	Parigi

[14] Cupellini Testori Vaggi "Il linguaggio SQL" FrancoAngeli

operandi siano espressi in un ordine corretto, ovvero prima il minore e poi il maggiore per cui un errore di sequenza porta a risultati scorretti.

IN

L'operatore IN verifica se un operando è contenuto in una sequenza di valori. Si può pervenire allo stesso risultato utilizzando l'operatore OR più volte ma l'operatore IN ha il pregio di rendere più concisa l'operazione

```
SELECT Nomep, Colore, Città
FROM P
WHERE colore IN ('rosso', 'verde')
Utilizzando OR si avrebbe avuto :
SELECT Nomep, Colore, Città
FROM P
WHERE Colore = 'rosso' OR Colore
='verde'
```

NomeP	Colore	Città
Dado	rosso	Londra
bullone	verde	Parigi
vite	rosso	Londra
dente	rosso	Londra

LIKE

L'operatore LIKE verifica se una stringa di caratteri corrisponde ad un determinato formato. Per definire il formato si utilizzano i caratteri jolly (_) e (%). Il primo identifica un singolo carattere mentre il secondo indica una sequenza di caratteri qualsiasi.

```
SELECT Nomep, Colore, Città
FROM P
WHERE LIKE 'ross_'

SELECT Nomep, Colore, Città
FROM P
WHERE Nomep LIKE 'd%'
```

NomeP	Colore	Città
dado	rosso	Londra
dente	rosso	Londra

IS NULL

L'operatore IS NULL verifica che il contenuto di un operando sia nullo. Può accadere che un elemento della tabella in una specifica colonna non contenga nessun valore

```
SELECT Nomep, Colore, Città
FROM P
WHERE peso IS NULL
```

Nessuna riga sarà selezionata.

Calcolo di espressioni

La lista selezionata dal comando SELECT non contiene solo nomi di colonne ma anche espressioni calcolate sui valori di una o più colonne. Nella valutazione di una espressione, se un operando è NULL allora il risultato della espressione è NULL, cioè non specificato.

'Prezzo unitario da $ a £

```
SELECT Categories.CategoryName, Categories.Description,
   Products.ProductName, Products.UnitPrice * 2180
FROM Categories INNER JOIN
   Products ON
   Categories.CategoryID = Products.CategoryID
WHERE (Categories.CategoryName = N'beverages')
```

Funzioni di gruppo

Le funzioni di gruppo (o di aggregazione) consentono di calcolare espressioni su insiemi di righe. Queste funzioni, quindi, permettono di

seguire calcoli sui valori di una colonna[15] . L'SQL ne prevede 5, ovvero MAX, MIN, SUM , AVG, COUNT[16].

MAX	La funzione MAX restituisce il valore massimo contenuto in una colonna	SELECT MAX (livello) 'Livello Max' FROM F	Livello Max 30
MIN	La funzione MIN restituisce il valore minimo contenuto in una colonna	SELECT MIN (livello) 'Livello Min' FROM F	Livello Min 10
SUM	La funzione SUN calcola la somma dei valori di una colonna. Si specificherà tramite DISTINCT se i valori sommati devono essere distinti o no. Il default è ALL.	SELECT SUM (livello) 'Tot. Livello' FROM F	Tot. Livello 110
AVG	La funzione AVG calcola la media aritmetica (average) dei valori non nulli di una colonna. I valori nulli (NULL) nel calcolo AVG non vengono semplicemente considerati	SELECT AVG (livello) 'Livello Medio' FROM F	Livello medio 22
COUNT	La funzione COUNT determina il numero dei valori non nulli contenuti in una colonna o il numero di righe di una tabella. Tramite le specifiche ALL e DISTINCT è possibile contare rispettivamente il numero di valori o righe selezionati o dei valori distinti (e non nulli).	SELECT COUNT (*) 'Numero Righe' FROM F	Numero di righe 5

```
SELECT MAX(Products.UnitPrice * 2180)
FROM Categories INNER JOIN
   Products ON
   Categories.CategoryID = Products.CategoryID
WHERE (Categories.CategoryName = 'beverages');


SELECT MIN(Products.UnitPrice * 2180)
FROM Categories INNER JOIN
   Products ON
   Categories.CategoryID = Products.CategoryID
WHERE (Categories.CategoryName = 'beverages');


SELECT AVG(Products.UnitPrice * 2180)
FROM Categories INNER JOIN
   Products ON
   Categories.CategoryID = Products.CategoryID
WHERE (Categories.CategoryName = N'beverages');

SELECT SUM(Products.UnitPrice * 2180)
FROM Categories INNER JOIN
   Products ON
   Categories.CategoryID = Products.CategoryID
WHERE (Categories.CategoryName = N'beverages');
```

[15] La funzione COUNT(*) opera su tutta la riga e non solo su di una colonna.

[16] "Le funzioni di aggregazione possono includere il termine DISTINCT, in tal caso vengono eliminati i valori duplicati. La presenza del termine DISTINCT con le funzioni max e min non influisce sull'elaborazione, in quanto la presenza di valori duplicati non incide sul valore minimo e massimo" Belski, Breschi, Pigni, Zocchi "dati e basi di dati : Il modello relazionale" - FrancoAngeli.

```
SELECT COUNT(Products.UnitPrice * 2180)
FROM Categories INNER JOIN
   Products ON
   Categories.CategoryID = Products.CategoryID
WHERE (Categories.CategoryName = N'beverages');
```

```
-- GROUP BY

  SELECT Categories.CategoryName,
     COUNT(Products.UnitPrice) AS Conta
  FROM Categories INNER JOIN
     Products ON
     Categories.CategoryID = Products.CategoryID
  GROUP BY Categories.CategoryName;
```

Proposizione HAVING

Specifica quali dei record raggruppati verranno visualizzati in un'istruzione SELECT con una proposizione GROUP BY. Dopo aver raggruppato i record con la proposizione GROUP BY, la proposizione HAVING visualizzerà i record che soddisfano le condizioni specificate.

La proposizione HAVING è simile alla proposizione WHERE che determina i record che verranno selezionati. Dopo aver raggruppato i record con la proposizione GROUP BY, la proposizione HAVING determinerà i record che verranno visualizzati.

La proposizione HAVING è facoltativa. Una proposizione HAVING può contenere fino a 40 espressioni collegate da operatori logici, quali ad esempio AND e OR.

```
SELECT Categories.CategoryName, COUNT(Products.UnitPrice)
   AS Conta
FROM Categories INNER JOIN
   Products ON
   Categories.CategoryID = Products.CategoryID
GROUP BY Categories.CategoryName
HAVING COUNT(Products.UnitPrice) > 10
```

Proposizione ORDER BY

La proposizione ORDER BY determina il tipo di ordinamento dei record recuperati nella query. Nella proposizione ORDER BY è possibile specificare le colonne che si desidera utilizzare come chiave di ordinamento e quindi indicare se i record verranno disposti in ordine crescente o decrescente. Nell'esempio che segue vengono restituiti tutti i record della tabella `DBO.EMPLOYEES` elencati per cognome in ordine alfabetico:

```
SELECT *
FROM DBO.EMPLOYEES
ORDER BY LASTNAME ASC;
```

In questo esempio la parola chiave ASC, che indica l'ordine crescente, è facoltativa. Il tipo di ordinamento crescente, ovvero da A a Z e da 0 a 9, è infatti l'impostazione predefinita. Per documentare chiaramente la proposizione ORDER BY, è tuttavia possibile specificare la parola chiave ASC dopo ciascun campo che si desidera disporre in ordine crescente.

Negli esempi che seguono, i nomi dei dipendenti vengono ordinati per cognome:

```
SELECT LASTNAME, FIRSTNAME
FROM DBO.EMPLOYEES
ORDER BY LASTNAME;
```

Per specificare l'ordine decrescente, ovvero da Z ad A e da 9 a 0, è necessario aggiungere la parola chiave DESC dopo ciascun campo che si desidera disporre in ordine decrescente.

```
SELECT LASTNAME, FIRSTNAME
FROM DBO.EMPLOYEES
ORDER BY LASTNAME DESC;
```

Anziché digitare nuovamente il nome della colonna di ordinamento, è possibile utilizzare il numero corrispondente alla posizione nell'elenco dell'istruzione SELECT:

```
SELECT LASTNAME, FIRSTNAME
FROM DBO.EMPLOYEES
ORDER BY 1 DESC;
```

È possibile includere altri campi nella proposizione ORDER BY. I record verranno innanzitutto ordinati in base al primo campo elencato dopo la proposizione ORDER BY. I record con valori uguali in tale campo verranno quindi ordinati in base al valore del secondo campo elencato e così via. Nell'esempio che segue, le retribuzioni vengono selezionate e quindi disposte in ordine decrescente. Tutti gli impiegati con retribuzione uguale vengono elencati in ordine alfabetico crescente:

```
SELECT Categories.CategoryName, Products.ProductName
FROM Categories INNER JOIN
    Products ON
    Categories.CategoryID = Products.CategoryID
ORDER BY Categories.CategoryName, Products.ProductName DESC;
```

Utilizzo di query di eliminazione

L'istruzione DELETE consente di creare una query per l'eliminazione di record da una o più tabelle elencate nella proposizione FROM e corrispondenti ai criteri della proposizione WHERE, come illustrato dal seguente esempio di sintassi:

DELETE
FROM *espressionetabella*
WHERE *criteri*

```
DELETE
FROM [Order Details]
WHERE OrderID=10248;
```

L'istruzione DELETE risulta particolarmente utile per l'eliminazione di più record.

Se si desidera eliminare tutti i record di una tabella, è consigliabile eliminare l'intera tabella anziché eseguire una query di eliminazione. L'eliminazione della tabella comporta, tuttavia, la perdita dell'intera struttura. Se invece si utilizza l'istruzione DELETE, verranno eliminati solo i dati, lasciando invariate la struttura e le proprietà della tabella, ad esempio gli indici e gli attributi dei campi.

L'istruzione DELETE consente di rimuovere record da una tabella singola oppure dalla parte "molti" di una relazione uno-a-molti. Una query di eliminazione consente di eliminare interi record e non i dati contenuti in un campo specifico. Per eliminare i valori di un campo, è necessario creare una query di aggiornamento che ne modifichi i valori in Null.

Non è possibile annullare un'operazione di rimozione di record eseguita utilizzando una query di eliminazione. Per verificare quali record sono stati eliminati, è innanzitutto necessario esaminare i risultati di una query di selezione che utilizzi gli stessi criteri e quindi eseguire la query di eliminazione.

È possibile creare in qualsiasi momento copie di backup dei dati in modo da poter recuperare eventuali record eliminati inavvertitamente.

Utilizzo di query di accodamento

L'istruzione INSERT INTO consente di aggiungere record a una tabella o di creare una query di accodamento.

Per eseguire una query di accodamento a record multipli, è possibile utilizzare la seguente sintassi:

INSERT INTO *destinazione*
SELECT [*origine*.]*campo1*[, *campo2*[, ...]
FROM *espressionetabella*

Per eseguire una query di accodamento a record singolo, è possibile utilizzare la seguente sintassi:

INSERT INTO *destinazione* [(*campo1*[, *campo2*[, ...]])]
VALUES (*valore1*[, *valore2*[, ...])

Per aggiungere un solo record a una tabella, è possibile utilizzare l'istruzione INSERT INTO con la sintassi della query di accodamento a record singolo. In questo caso, il codice specificherà il nome e il valore di ciascun campo del record. È necessario specificare tutti i campi del record a cui si desidera assegnare un valore, nonché il valore desiderato per ciascun campo. Se non vengono specificati tutti i campi, nelle colonne mancanti verrà inserito il valore predefinito oppure il valore Null. I record verranno aggiunti alla fine della tabella.

L'istruzione INSERT INTO consente inoltre di aggiungere un gruppo di record contenuti in un'altra tabella o query utilizzando la proposizione SELECT...FROM come indicato nella sintassi per la query di accodamento a record multipli. In questo caso, la proposizione SELECT specifica i campi che verranno aggiunti alla tabella di destinazione specificata dall'argomento *destinazione*.

La tabella di origine o di destinazione può specificare una tabella o una query. Se viene specificata una query, il modulo di gestione di database aggiunge un gruppo di record a tutte le tabelle specificate dalla query.

L'istruzione INSERT INTO viene in genere utilizzata per aggiungere una nuova tabella di record relativi a clienti nella tabella Clienti attiva.

L'istruzione INSERT INTO è facoltativa. Quando viene inclusa, deve precedere l'istruzione SELECT.

Per verificare quali record verranno aggiunti prima di eseguire la query di accodamento, è innanzitutto necessario eseguire e visualizzare i risultati di una query di selezione che utilizzi gli stessi criteri.

Una query di accodamento consente di copiare record tra tabelle e non produrrà alcun effetto sulle tabelle contenenti i record aggiunti.

Anziché aggiungere record esistenti in un'altra tabella, è possibile specificare il valore di ciascun campo di un nuovo record singolo utilizzando la proposizione VALUES. Se l'elenco dei campi viene omesso, nella proposizione VALUES è necessario includere un valore per ciascun campo

della tabella. In caso contrario l'istruzione INSERT non verrà eseguita. È necessario utilizzare un'istruzione INSERT INTO con una proposizione VALUES per ciascun record aggiuntivo che si desidera creare.

```
CREATE TABLE LAVORATORE
      (CODLAV VARCHAR(10),
     CODSTRUTTURA VARCHAR(10),
     NOME VARCHAR(20),
     COGNOME VARCHAR(20),
     CODQUAL VARCHAR(10),
     CODSEDEPOSTA VARCHAR(10),
     DESCRSEDEPOSTA VARCHAR(100),
     INDIRIZZO VARCHAR(150),
     COMUNE VARCHAR(30),
     CAP VARCHAR(5),
     PROVINCIA VARCHAR (2),
     FAX VARCHAR(14),
     TEL VARCHAR(14),
     POSTA VARCHAR(100),
     PRIMARY KEY (CODLAV)
     );


INSERT INTO LAVORATORE VALUES
('079116','00480','Mario','Rossi','QF07','005','UFF. SPEC. RAPPORTI CON
AZIENDE OSPED.','P.ZA S.MARCO, 4','FIRENZE','50100','FI',null,null,null);

INSERT INTO LAVORATORE VALUES
('092962','50800','MASSIMO','INNOCENTE','QF08','610','DIP. DI CHIMICA','VIA
G.CAPPONI, 9','FIRENZE','50100','FI',null,null,null);

INSERT INTO LAVORATORE VALUES
('035958','52900','CRISTINA','LETMI','QF06','685','DIP. SCIENZE
FISIOLOGICHE','V.LE MORGAGNI, 63','FIRENZE','50100','FI',null,null,null);

INSERT INTO LAVORATORE VALUES
('096577','00755','MANOLA','LEONARDONI','QF03','696','DIP. DI SCIENZE
BIOCHIMICHE','V.LE MORGAGNI, 50','FIRENZE','50100','FI',null,null,null);

INSERT INTO LAVORATORE VALUES
('036399','52400','ROSSELLA','HOARA','QF06','660','DIP. BIOLOGIA ANIMALE E
GENETICA','VIA ROMANA, 17','FIRENZE','50100','FI',null,null,null);

INSERT INTO LAVORATORE VALUES
('093408','00380','MARIA','MARADEL','QF06','006','UFFICIO SEGRETERIE
STUDENTI','P.ZA S.MARCO, 4','FIRENZE','50100','FI',null,null,null);

INSERT INTO LAVORATORE VALUES
('078856','52300','PATRIZIA','MARONGHI','QF06','655','DIP. CHIMICA
ORGANICA','VIA G.CAPPONI, 9','FIRENZE','50100','FI',null,null,null);

INSERT INTO LAVORATORE VALUES
('082978','00780','GIULIA','MORONGI','QF06','201','POLO6 BIBL.DI
MEDICINA','V.LE MORGAGNI, 85','FIRENZE','50100','FI',null,null,null);


CREATE TABLE LAVORATORE2
      (CODLAV VARCHAR(10),
     CODSTRUTTURA VARCHAR(10),
     NOME VARCHAR(20),
     COGNOME VARCHAR(20),
```

```
    CODQUAL VARCHAR(10),
    CODSEDEPOSTA VARCHAR(10),
    DESCRSEDEPOSTA VARCHAR(100),
    INDIRIZZO VARCHAR(150),
    COMUNE VARCHAR(30),
    CAP VARCHAR(5),
    PROVINCIA VARCHAR (2),
    FAX VARCHAR(14),
    TEL VARCHAR(14),
    POSTA VARCHAR(100),
    PRIMARY KEY (CODLAV)
    );
```

```
INSERT INTO LAVORATORE2
SELECT *
FROM LAVORATORE;
```

Non droppare le tabelle Lavoratore,Lavoratore2

Utilizzo di Query di aggiornamento

L'istruzione UPDATE consente di creare una aggiornamento per modificare i valori dei campi della tabella specificata in base ai criteri specificati.

UPDATE *tabella*
SET *nuovovalore*
WHERE *criteri*;

L'istruzione UPDATE risulta particolarmente utile se si desidera modificare numerosi record oppure quando i record da modificare sono contenuti in più tabelle

```
UPDATE LAVORATORE
SET COGNOME = 'Deficente'
WHERE (COGNOME = 'Innocente')
```

L'istruzione UPDATE non genera risultati. Per verificare quali record verranno modificati, è necessario esaminare i risultati di una SELECT che utilizzi gli stessi criteri prima di eseguire la selezione di aggiornamento.

Join

Una delle caratteristiche più valide dei database relazionali è la possibilità di unire due o più tabelle per creare una nuova tabella o gruppo di record contenente le informazioni di entrambe le tabelle.

Le tabelle vengono unite in base alla relazione esistente tra di esse, in genere tra la chiave primaria di una tabella e la corrispondente chiave esterna dell'altra tabella. A seconda del modo in cui le tabelle vengono unite, è possibile creare i seguenti tipi di join:

INNER JOIN: Join interno. Nel join vengono inclusi i record di entrambe le tabelle solo se il campo specificato della prima tabella corrisponde al campo specificato nella seconda.

LEFT JOIN: Join esterno sinistro. Nel join vengono inclusi tutti i record della prima tabella e i record della seconda in cui i campi specificati corrispondono.

RIGHT JOIN: Join esterno destro. Nel join vengono inclusi tutti i record della seconda tabella e i record della prima in cui i campi specificati corrispondono.

Join interno

Per creare una query che includa solo i record contenenti gli stessi dati nei campi uniti, è possibile utilizzare un'operazione INNER JOIN.

L'operazione INNER JOIN consente di combinare i record di due tabelle quando vi sono dati corrispondenti in un campo comune. Utilizzare la sintassi riportata di seguito:

FROM *tabella***,** *tabella2*
WHERE *tabella1.campo1* **=** *tabella2.campo2*

Verrà creato un join con abbinamento, noto anche come join interno. I join con abbinamento sono i join più comuni e consentono di combinare i record di due tabelle quando vi sono dati corrispondenti in un campo comune.

```
SELECT Categories.CategoryName, Products.ProductName,
   Products.UnitPrice
FROM Categories, Products
WHERE Categories.Categoryid = Products.Categoryid
ORDER BY Categories.CategoryName
```

Un Modo alternativo di realizzare il Join è quello di usare la sintassi **Join**

FROM *tabella1* **INNER JOIN** *tabella2* **ON** *tabella1.campo1* **=** *tabella2.campo2*

È possibile utilizzare un'operazione INNER JOIN in qualsiasi proposizione FROM. Verrà creato un join con abbinamento.

```
SELECT Categories.CategoryName, Products.ProductName,
   Products.UnitPrice
FROM Categories INNER JOIN
   Products ON
   Categories.CategoryID = Products.CategoryID
ORDER BY Categories.CategoryName
```

Join esterno (sinistro, destro)

I join esterni combinano i record della tabella di origine indicati in una proposizione FROM e utilizzano la seguente sintassi:

FROM *tabella1* [**LEFT** | **RIGHT**]**OUTER JOIN** *tabella2*
ON *tabella1.campo1 = tabella2.campo2*

Utilizzare un'operazione LEFT OUTER JOIN per creare un join esterno sinistro che consente di includere tutti i record della prima di due tabelle, ovvero quella a sinistra, anche se la seconda tabella, ovvero quella a destra, non contiene record con valori corrispondenti.

Utilizzare un'operazione RIGHT OUTER JOIN per creare un join esterno destro che consente di includere tutti i record della seconda di due tabelle, ovvero quella a destra, anche se la prima tabella, ovvero quella a sinistra, non contiene record con valori corrispondenti.

.

Nell'esempio che segue vengono illustrate le modalità di unione delle tabelle Categories e Products in base al campo CategoryID. La query produce l'elenco di tutte le categorie, comprese quelle che non contengono prodotti:

```
Inserire una Categoria di prodotti in Categories ed eseguire la query

SELECT Categories.CategoryName, Products.ProductName,
    Products.UnitPrice
FROM Categories LEFT OUTER JOIN
    Products ON
    Categories.CategoryID = Products.CategoryID
ORDER BY Categories.CategoryName;
```

In questo esempio CategoryID è il campo unito, che tuttavia non essendo incluso nell'istruzione SELECT non viene incluso nemmeno nel risultato della query. Per includere il campo unito, è necessario specificare il nome del campo nell'istruzione SELECT, in questo caso `Categories.[CategoryID]`.

Join nidificati

È inoltre possibile nidificare le istruzioni JOIN utilizzando la seguente sintassi:

SELECT *campi*
FROM *tabella1* **INNER JOIN**
(*tabella2* **INNER JOIN** [( ]*tabella3*
[**INNER JOIN** [( ]*tabellax* [**INNER JOIN** ...)]
ON *tabella3.campo3* = *tabellax.campox*)]
ON *tabella2.campo2* = *tabella3.campo3*)
ON *tabella1.campo1* = *tabella2.campo2*;

OUTER JOIN:

Left Join

Union

Right Join

È possibile unire campi numerici di qualsiasi tipo anche se contengono tipi di dati diversi. È possibile, ad esempio, unire un campo Contatore e un campo Numero la cui proprietà Size dell'oggetto Field sia impostata su Integer.

Interrogazione di unione

È possibile utilizzare l'operazione UNION per creare una query di unione che combini i risultati di due o più query o tabelle indipendenti.

query1 **UNION** [**ALL**] *query2* [**UNION** [**ALL**] *queryn* [...]]

Il segnaposto *query* rappresenta un'espressione stringa che identifica il campo contenente i dati numerici di cui si desidera calcolare la media oppure un'espressione che esegue un calcolo utilizzando i dati del campo. Gli operandi nell'espressione possono includere il nome di un campo

della tabella, una costante o una funzione, che può essere intrinseca oppure definita dall'utente, ma non una delle altre funzioni di aggregazione SQL o di dominio.

È possibile unire i risultati di una query e di un'istruzione SQL in una singola operazione UNION. Nell'esempio che segue vengono uniti i risultati di una query esistente, denominata Nuovi conti, e di un'istruzione SELECT:

```
SELECT *
FROM Lavoratore
UNION ALL
SELECT *
FROM Lavoratore2
where Lavoratore2.Codqual='QF06';
```

Per impostazione predefinita, quando si utilizza un'operazione UNION non verranno restituiti record duplicati. È tuttavia possibile includere il predicato ALL per fare in modo che vengano restituiti tutti i record. In questo modo l'interrogazione verrà eseguita più rapidamente.

Tutte le interrogazioni di un'operazione UNION devono richiedere lo stesso numero di campi, che tuttavia possono essere di dimensioni diverse e contenere tipi di dati diversi.

È possibile utilizzare una proposizione GROUP BY e/o HAVING in ciascun argomento *query* per raggruppare i dati restituiti. È inoltre possibile utilizzare una proposizione ORDER BY alla fine dell'ultimo argomento *interrogazioni* per visualizzare i dati restituiti nell'ordine specificato.

Ottimizzazione delle interrogazioni

L'ottimizzazione consiste in una serie di operazioni che interessano diversi fattori estranei all'esecuzione delle interrogazioni, ma che incidono ugualmente sulle prestazioni del database, ad esempio, la configurazione del software e dell'hardware, l'installazione di Windows e le dimensioni e la cache del disco.

Strategie generali di ottimizzazione

- Evitare un utilizzo eccessivo dell'ordinamento dei campi, in particolare per i campi non indicizzati.
- Verificare che le tabelle che vengono unite da origini diverse siano indicizzate oppure vengano unite in base alla chiave primaria.
- Utilizzare gli operatori BETWEEN...AND, IN e = per le colonne indicizzate.
- Quando si crea una interrogazione, evitare di aggiungervi campi non necessari.

Le viste

Le viste rappresentano il terzo livello (livello esterno) dei database. Nei RDBMS server sono i corrispettivi delle query di Access. Non possono contenere parametri.

Creazione delle viste

Si usa i l comando CREATE VIEW. Nel comando CREATE VIEW si possono specificare diverse opzioni, tra cui:

- La query che definisce la vista
- Gli alias per le colonne della vista

CREATE VIEW *view_name* [WITH ENCRYPTION] AS
SELECT *statement...*
FROM *table_name* | *view_name*

```
CREATE VIEW NomeImp AS
SELECT COGNOME,NOME
FROM LAVORATORE
GO
```

In questo caso si è create una vista che riprodice tutto il contenuto della tabella Lavoratore. Posso d'ora in poi utilizzare la vista come se fosse una tabella.
Posso, ad esempio, aggiungere una riga

```
INSERT INTO LAVORATORE VALUES ('043403','00950','MARIA
CRISTINA','ROMANO','QF08','900','CENTRO DI CALCOLO ELETTRONICO','VIA DELLE
GORE, 2','FIRENZE','50100','FI',null,null,null);

INSERT INTO LAVORATORE VALUES
('073451','53800','PAOLA','RICCI','QF06','691','DIP.SCIENZA DEL SUOLO E
NUTRIZ.D.PIANTA','P.LE DELLE
CASCINE','FIRENZE','50100','FI',null,null,null);

'Creo una View che contiene gli stessi campi della tabella impiegati

CREATE VIEW IMPIEGATI AS
SELECT *
FROM LAVORATORE
Go


INSERT INTO Impiegati VALUES
('025357','00755','PEA','PERICOLI','QF05','274','SETT.TEC.LOG. C/O
CE.SIT.','CE.SIT. VIA DELLE GORE,
2','FIRENZE','50100','FI',null,null,null);
```

Oppure creare un Join:

```
Creare La tabella Strutture (controllare se è già esistente)
CREATE TABLE STRUTTURA
      (
      CODSTRUTTURA VARCHAR2(10),
      DENOMINAZIONE VARCHAR2(150),
      INDIRIZZO VARCHAR2(150),
      CAP VARCHAR2 (5),
      COMUNE VARCHAR2 (30),
      PROVINCIA VARCHAR2 (2),
      FAX VARCHAR2(14),
      TEL VARCHAR2(14),
      ANNOTA VARCHAR2 (255),
      PRIMARY KEY (CODSTRUTTURA)
      );

     Inserire Delle righe:

INSERT INTO STRUTTURA VALUES ('00950','CENTRO SERVIZI INFORMATICI E
TELEMATICI',null,null,null,null,null,null,null);

INSERT INTO STRUTTURA VALUES ('00951','SEZIONE TECNICA DEI SERVIZI
AMMINISTRATIVI',null,null,null,null,null,null,null);

INSERT INTO STRUTTURA VALUES ('00952','SEZIONE SUPPORTO FUNZIONALE E
RICERCA',null,null,null,null,null,null,null);
```

```
INSERT INTO STRUTTURA VALUES ('00953','SEZIONE ANALISI DEI
SISTEMI',null,null,null,null,null,null,null);
```

```
CREATE VIEW ImpAfferenza as
SELECT  IMPIEGATI.COGNOME,STRUTTURA.DENOMINAZIONE AS  AFFERENZA
FROM IMPIEGATI,STRUTTURA
WHERE IMPIEGATI.CODSTRUTTURA=STRUTTURA.CODSTRUTTURA
GO
```

```
select * from ImpAfferenza;

COGNOME              AFFERENZA
-------------------- ----------------------------------------------------------------------------------------------------
ROMANO               CENTRO SERVIZI INFORMATICI E TELEMATICI

(1 row(s) affected)
```

Oppure creare dei set di dati più ridotti:

```
CREATE VIEW ImpiegatiB as
Select * from Lavoratore
Where Cognome like 'R%'
GO

CREATE VIEW ImpiegatiBNC as
Select Cognome,Nome from Lavoratore
Where Cognome like 'R%'
GO
```

Le modalità di creazione di una vista influenzano direttamente le funzionalità della vista stessa. Una Vista come Impiegati può fare tutte le operazioni DML comprese SELECT,INSERT,UPDATE e DELETE. Mentre una Vista complesa come ImpiegatiBNC ha delle limitazioni poiché non è chiaro come operare sulla tabelle di origine.
Le viste complesse sono quelle che hanno join, operatori come DISTINCT o funzioni di gruppo come Group By.

Alias come nomi di colonna di una vista

E' possibile attribuire alle colonne di una vista degli alias rispetto ai nomi delle colonne della o delle tabelle di origine.

```
CREATE VIEW Impiegati2 (Last_Name,First_Name) as
Select Cognome,Nome
from Lavoratore
GO
```

Eliminazione Viste

```
DROP VIEW Impiegati2;
```

Funzioni Di T-SQL

Funzioni legate alle stringhe

Concatenazione (+)
Serve per concatenare le stringhe in una select

select cognome +' '+ nome as nominativo
from lavoratore;

```
nominativo
-------------------------------------------
PERICOLI PEA
LETMI CRISTINA
HOARA ROSSELLA
ROMANO MARIA CRISTINA
RICCI PAOLA
MARONGHI PATRIZIA
Rossi Mario
MORONGI GIULIA
INNOCENTE MASSIMO
MARADEL MARIA
LEONARDONI MANOLA

(11 row(s) affected)
```

Ltrim,Rtrim

Eliminano gli spazi esterni sinistri o destri della stringa

Select Ltrim(' AAA '+ cognome+' ') as Cognome2
from Lavoratore;

```
Cognome2
-------------------------------------
AAA  PERICOLI
AAA  LETMI
AAA  HOARA
AAA  ROMANO
AAA  RICCI
AAA  MARONGHI
AAA  Rossi
AAA  MORONGI
AAA  INNOCENTE
AAA  MARADEL
AAA  LEONARDONI

(11 row(s) affected)
```

Select Rtrim(' AAA '+ cognome+' ') as Cognome2
from Lavoratore;

```
Cognome2
-------------------------------------
    AAA  PERICOLI
    AAA  LETMI
    AAA  HOARA
    AAA  ROMANO
    AAA  RICCI
    AAA  MARONGHI
    AAA  Rossi
    AAA  MORONGI
    AAA  INNOCENTE
    AAA  MARADEL
    AAA  LEONARDONI

(11 row(s) affected)
```

'il Trim destro e sinistro si effettua così

Select LTRim(Rtrim(' AAA '+ cognome+' ')) as Cognome2
from Lavoratore;

```
Cognome2
------------------------------------
AAA   PERICOLI
AAA   LETMI
AAA   HOARA
AAA   ROMANO
AAA   RICCI
AAA   MARONGHI
AAA   Rossi
AAA   MORONGI
AAA   INNOCENTE
AAA   MARADEL
AAA   LEONARDONI

(11 row(s) affected)
```

Upper

Trasforma tutte le lettere in maiuscolo

Select Upper(cognome) from Lavoratore;

Lower

Trasforma tutte le lettere in minuscolo
Select lower(cognome) from Lavoratore;

len
Riporta la lunghezza della stringa

Select len(cognome) from Lavoratore;

SUBSTRING

Seleziona parte delle stringhe

SUBSTR(stringa,inizio[,Conta])

Select cognome, substring(cognome,1,3) as PrimiTre
from Lavoratore;

```
cognome              PrimiTre
-------------------- --------
PERICOLI             PER
LETMI                LET
HOARA                HOA
ROMANO               ROM
RICCI                RIC
MARONGHI             MAR
Rossi                Ros
MORONGI              MOR
INNOCENTE            INN
MARADEL              MAR
LEONARDONI           LEO

(11 row(s) affected)
```

CHARINDEX
Restituisce il numero della posizione di una stringa cercata

Select cognome, CharIndex('C',cognome) as MyPos
from Lavoratore;

```
cognome              MyPos
-------------------- -----------
PERICOLI             5
LETMI                0
HOARA                0
```

```
ROMANO              0
RICCI               3
MARONGHI            0
Rossi               0
MORONGI             0
INNOCENTE           5
MARADEL             0
LEONARDONI          0

(11 row(s) affected)
```

REVERSE

Inverte l'ordine delle lettere in una stringa

Select cognome, reverse(cognome) as Inverto
from Lavoratore;

```
cognome              Inverto
-------------------- --------------------
PERICOLI             ILOCIREP
LETMI                IMTEL
HOARA                ARAOH
ROMANO               ONAMOR
RICCI                ICCIR
MARONGHI             IHGNORAM
Rossi                issoR
MORONGI              IGNOROM
INNOCENTE            ETNECONNI
MARADEL              LEDARAM
LEONARDONI           INODRANOEL

(11 row(s) affected)
```

REPLICATE

Replica una stringa n volte

Select cognome,replicate(cognome,2) as replico
from Lavoratore;

```
cognome              replico
-------------------- ----------------------------------------
PERICOLI             PERICOLIPERICOLI
LETMI                LETMILETMI
HOARA                HOARAHOARA
ROMANO               ROMANOROMANO
RICCI                RICCIRICCI
MARONGHI             MARONGHIMARONGHI
Rossi                RossiRossi
MORONGI              MORONGIMORONGI
INNOCENTE            INNOCENTEINNOCENTE
MARADEL              MARADELMARADEL
LEONARDONI           LEONARDONILEONARDONI

(11 row(s) affected)
```

RIGHT-LEFT

Restituisce la parte destra/sinistra di una stringa secondo il numero dei caratteri specificati

Select cognome,right(cognome,4) as parteDestra
from Lavoratore;

```
cognome              parteDestra
```

```
------------------- -----------
PERICOLI            COLI
LETMI               ETMI
HOARA               OARA
ROMANO              MANO
RICCI               ICCI
MARONGHI            NGHI
Rossi               ossi
MORONGI             ONGI
INNOCENTE           ENTE
MARADEL             ADEL
LEONARDONI          DONI

(11 row(s) affected)
```

Funzioni legate ai numeri

Transact-SQL Mathematical Functions

Funzione	*Parametro*	*Restituisce*
ACOS	*(float_expression)*	*Angle in radians whose cosine is a FLOAT value.*
ASIN	*(float_expression)*	*Angle in radians whose sine is a FLOAT value.*
ATAN	*(float_expression)*	*Angle in radians whose tangent is a FLOAT value.*
ATAN2	*(float_expr1,float_expr2)*	*Angle in radians whose tangent is float_expr1/floatexpr2.*
COS	*(float_expression)*	*Trigonometric cosine of angle in radians.*
COT	*(float_expression)*	*Trigonometric cotangent of angle in radians.*
SIN	*(float_expression)*	*Trigonometric sine of angle in radians.*
TAN	*(float_expression)*	*Trigonometric tangent of expression in radians.*
DEGREES	*(numeric_expression)*	*Degrees converted from radians returned as the same datatype as expression. Datatypes can be INTEGER, MONEY, REAL and FLOAT.*
RADIANS	*(numeric_expression)*	*Radians converted from degrees returned as the same datatype as expression. Datatypes can be INTEGER, MONEY, REAL, and FLOAT.*
CEILING	*(numeric_expression)*	Produce il massimo valore intero >= di quello specificato Ceil(2)=2; Ceil(2,4)=3; Ceil(-2.7)=-3
FLOOR	*(numeric_expression)*	Opposto di ceil: Produce il minimo valore intero <= di quello specificato Floor(2)=2; Floor(2.4)=2; Floor(-2.7)=2
EXP	*(float_expression)*	Valore esponenziale
LOG	*(float_expression)*	Logaritmo naturale
LOG10	*(float_expression)*	Logaritmi in base 10
PI()		PI Greco= 3.1415926535897936.
POWER	*(numeric_expression,y)*	Elevazione a potenza Power(valore,esponente) Power(3,3)=27; Power(3,2)=9
ABS	*(numeric_expression)*	Valore Assoluto
RAND	*([integer_expression])*	Restituisce un Valore Random
ROUND	*(numeric_expr,integer_expr)*	Arrotonda a partire dal valore intero corrispondente alla posizione del decimale
SIGN	*(numeric_expression)*	*One, zero, or –1 returned as the same datatype as expression. Datatypes can be INTEGER, MONEY, REAL, and FLOAT.*
SQRT	*(float_expression)*	Radice quadrata

Funzioni legate alle date

GETDATE()

Ora corrente.

select Getdate ();

DATENAME-DATEPART

DATENAME restituisce una parte della data con valore string

DATENAME(*<date part>*, *<date>*)
DATEPART(*<date part>*, *<date>*)

select GETDATE(),datename(month,birthdate)
from employees;

```
--------------------------- ----------------
2001-03-27 11:53:52.243     December
2001-03-27 11:53:52.243     February
2001-03-27 11:53:52.243     August
2001-03-27 11:53:52.243     September
2001-03-27 11:53:52.243     March
2001-03-27 11:53:52.243     July
2001-03-27 11:53:52.243     May
2001-03-27 11:53:52.243     January
2001-03-27 11:53:52.243     January

(9 row(s) affected)
```

Date part	**Abbreviazione**
year	yy, yyyy
quarter	qq, q
month	mm, m
dayofyear	dy, y
day	dd, d
week	wk, ww
weekday	dw
hour	hh
minute	mi, n
second	ss, s
millisecond	ms

DATEADD

Incrementa o decrementa una data

DATEADD (*<date part>*, *<number>*, *<date>*)

select GETDATE()as Adesso,dateadd(year,5,GETDATE()) as incrementoAnno,dateadd(year,-5,GETDATE()) as decrementoAnno
from employees;

```
Adesso                      incrementoAnno              decrementoAnno
--------------------------- --------------------------- ---------------------------
2001-03-27 12:01:56.230     2006-03-27 12:01:56.230     1996-03-27 12:01:56.230
2001-03-27 12:01:56.230     2006-03-27 12:01:56.230     1996-03-27 12:01:56.230
2001-03-27 12:01:56.230     2006-03-27 12:01:56.230     1996-03-27 12:01:56.230
2001-03-27 12:01:56.230     2006-03-27 12:01:56.230     1996-03-27 12:01:56.230
2001-03-27 12:01:56.230     2006-03-27 12:01:56.230     1996-03-27 12:01:56.230
2001-03-27 12:01:56.230     2006-03-27 12:01:56.230     1996-03-27 12:01:56.230
2001-03-27 12:01:56.230     2006-03-27 12:01:56.230     1996-03-27 12:01:56.230
2001-03-27 12:01:56.230     2006-03-27 12:01:56.230     1996-03-27 12:01:56.230
2001-03-27 12:01:56.230     2006-03-27 12:01:56.230     1996-03-27 12:01:56.230

(9 row(s) affected)
```

DATEDIFF
Calcola la differenza fra 2 date

DATEDIFF(<*date part*>, <*date1*>, <*date2*>)

select GETDATE()as Adesso,birthdate,datediff(year,birthdate,GETDATE()) as DifferenzaInAnni from employees;

```
Adesso                  birthdate                DifferenzaInAnni
----------------------- ------------------------ ---------------
2001-03-27 12:05:18.760 1948-12-08 00:00:00.000  53
2001-03-27 12:05:18.760 1952-02-19 00:00:00.000  49
2001-03-27 12:05:18.760 1963-08-30 00:00:00.000  38
2001-03-27 12:05:18.760 1937-09-19 00:00:00.000  64
2001-03-27 12:05:18.760 1955-03-04 00:00:00.000  46
2001-03-27 12:05:18.760 1963-07-02 00:00:00.000  38
2001-03-27 12:05:18.760 1960-05-29 00:00:00.000  41
2001-03-27 12:05:18.760 1958-01-09 00:00:00.000  43
2001-03-27 12:05:18.760 1966-01-27 00:00:00.000  35

(9 row(s) affected)
```

Date Parts Used in Date Functions

Date Part	***Abbreviazione***	***Valori***
Year	*yy*	*1753-9999*
Quarter	*qq*	*1-4*
Month	*mm*	*1-12*
Day of Year	*dy*	*1-366*
Day	*dd*	*1-31*
Week	*wk*	*1-54*
Weekday	*dw*	*1-7 (Sun-Sat)*
Hour	*hh*	*0-23*
Minute	*mi*	*0-59*
Second	*ss*	*0-59*
Millisecond	*ms*	*0-999*

Funzioni di conversione

CONVERT
Converte una espressione da un tipo di dato ad unaltro

CONVERT(<*datatype*> [(<*length*>)], <*expression*> [, <*style*>])

select birthdate,convert(char(20),birthdate,105) DaDatetimeAChar from employees;

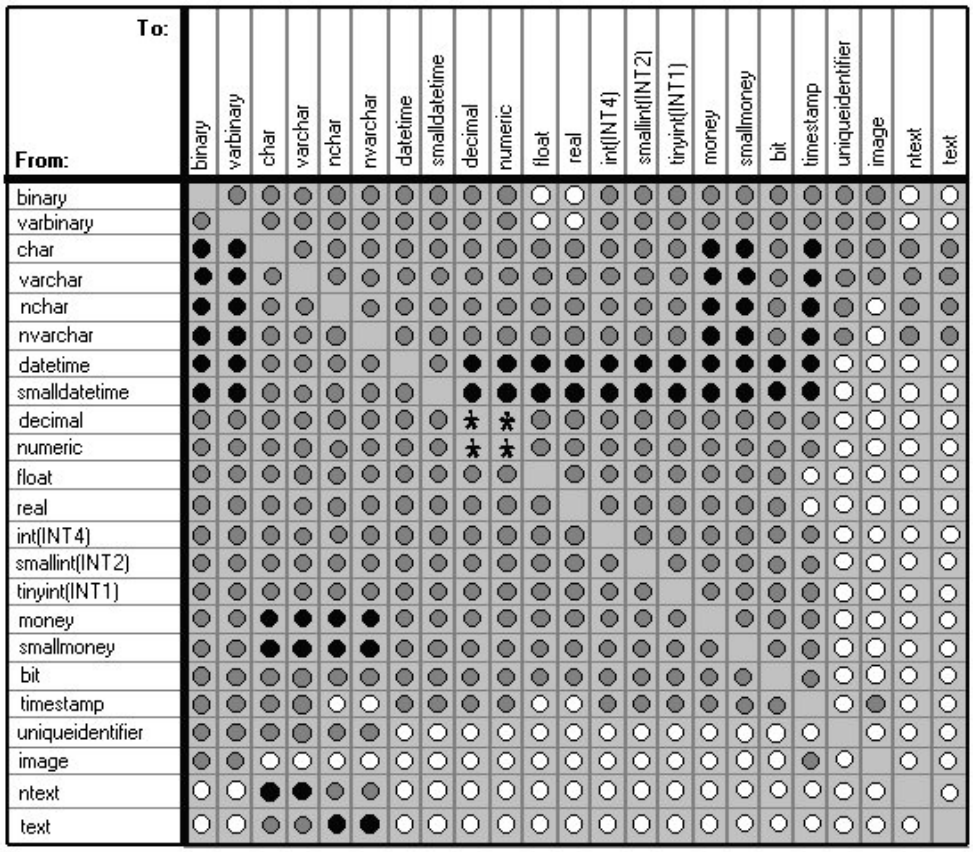
```
birthdate                DaDatetimeAChar
------------------------ --------------------
1948-12-08 00:00:00.000  08-12-1948
1952-02-19 00:00:00.000  19-02-1952
1963-08-30 00:00:00.000  30-08-1963
1937-09-19 00:00:00.000  19-09-1937
1955-03-04 00:00:00.000  04-03-1955
1963-07-02 00:00:00.000  02-07-1963
1960-05-29 00:00:00.000  29-05-1960
1958-01-09 00:00:00.000  09-01-1958
1966-01-27 00:00:00.000  27-01-1966

(9 row(s) affected)
```

Parametri per determinare lo Syle della data

Style Numbers for the `CONVERT` *Function*

Without	With Century (yy)	Century (yyyy)	Standard Display
-	0 or 100	default	mon dd yyyy hh:miAM(orPM)
1	101	USA	mm/dd/yy
2	102	ANSI	yy.mm.dd
3	103	English/French	dd/mm/yy
4	104	German	dd.mm.yy
5	105	Italian	dd-mm-yy
6	106		dd mon yy
7	107		mon dd, yy
8	108		hh:mi:ss
9	109		mon dd yyyy hh:mi:sssAM (orPM)
10	110	USA	mm-dd-yy
11	111	Japan	yy/mm/dd
12	112	ISO	yymmdd
13	113	Europe	dd mon yyyy hh:mi:ss:mmm (24h)
14	114	-	hh:mi:ss::mmm (24h)



Funzioni di sistema

System Functions

Function	Parameter(s)	Information Returned
HOST_NAME()		The name of the server computer
HOST_ID()		The ID number of the server computer
SUSER_ID	(['*login-name*'])	The login number of the user
SUSER_NAME	([*server_user_id*])	The login name of the user
USER_ID	(['*user_name*'])	The database ID number of the user
USER_NAME	([*user_id*])	The database username of the user
DB_NAME	(['*database_id*'])	The name of the database
DB_ID	(['*database_name*'])	The ID number of the database
GETANSINULL	(['*database_name*'])	Returns 1 for ANSI nullability, 0 if ANSI nullability not defined
OBJECT_ID	('*object_name*')	The number of a database object
OBJECT_NAME	(*object_id*)	The name of a database object
INDEX_COL	('*table_name*', *index_id*,	The name of the index *key_id)* column
COL_LENGTH	('*table_name*',	The defined length of a '*column_name*') column
COL_NAME	(*table_id*, *column_id*)	The name of the column
DATALENGTH	('*expression*')	The actual length of an expression of a datatype
IDENT_INCR	('*table_or_view*')	The increment (returned as numeric(@@MAXPRECISION,0)) for a column with

		the identity property
IDENT_SEED	('*table_or_view*')	The seed value (returned as numeric(@@MAXPRECISION,0)) for a column with the identity property
STATS_DATE	(*table_id*, *index_id*)	The date that the statistics for the index (*index_id*) were last updated
COALESCE	(*expression1*, *expression2*,... *expressionN*)	Returns the first non-null expression
ISNULL	(*expression*, *value*)	Substitutes value for each NULL entry
NULLIF	(*expression1*, *expression2*)	Returns a NULL when expression1 is NULL when expression1 is equivalent to expression2

Restituisce il nome dell'Host cui sono collegato

.

```
select host_name ()
------------------------------
NT1

(1 row(s) affected)


select host_name (),host_id (),db_name (), db_id (), suser_name ()
---------- -------- ---------- -------------------------
NT1        0000005e employees          6      sa

(1 row(s) affected)
```

Capitolo quarto

Rules e Stored Procedures

Le Regole (Rules)

Una regola provvede a definire delle restrizioni per valori da o per modificare valori sempre in nelle colonne del Database.Le fasi necessarie alla creazione di una regola in T-SQL consistono nel creare la regola per poi associarla a una o più colonne, in modo da attivare la regola stessa.
Esempio:
Una regola può essere associata ad una colonna in cui deve essere registrato il dipartimento di appartenenza di un dipendente.Se in una azienda esistono solo 4 dipartimenti si può con una regola limitare l'inserimento solo ai 4 dipartimenti respingendo gli inserimenti diversi.

Sintassi per la creazione di Rules:

```
CREATE RULE rule_name
AS condition_expression
```

Nell'espressione condizionale può essere usata una clausola WHERE oppure una comparazione aritmetica o fra più operatori.L'espressione condizionale che si usa in una regola deve essere preceduta dal simbolo @.Si usa @ per fare riferimento a parametri che devono essere utilizzati in operazioni come INSERT o UPDATE.

1) Creazione regola

```
'Creo colonna department
alter table employees add department varchar (100);

'quando si crea una regola non mettere ";" alla fine del blocco

create rule department_values
as @department in ('Sales','Field Service','Logistics','Software')
```

Si è creata una regola che ammette solo i valori riportati dentro l'operatore IN

2) **Associare la regola alla colonna di una tabella**
Occorre usare una procedura memorizzata (Stored Procedure) di sistema **sp_bindrule** per associare una regola ad una colonna.La procedura sp_bindrule seguie questa sintassi:

```
sp_bindrule rulename, table_name.column_name]
```

rulename=max 30 caratteri
table_name.column_name=indica il valore della colonna a cui associarlo

```
'associo alla colonna department
sp_bindrule department_values, 'employees.department'
```

Rule bound to table column.

```
insert into employees(Lastname,Firstname,department)
values ('Rossi','Mario','Amministrazione')
```

Server: Msg 513, Level 16, State 1, Line 1
A column insert or update conflicts with a rule imposed by a previous CREATE RULE statement. The statement was terminated. The conflict occurred in database 'Northwind', table 'Employees', column 'department'.
The statement has been terminated.

```
'riproviamo con un valore giusto permesso dalla regola
```

```
insert into employees(Lastname,Firstname,department)
values ('Rossi','Mario','Logistics')

(1 row(s) affected)
```

Altro Esempio

```
'Creo la regola RUL_LowerCase che accetta solo lettere minuscole

CREATE RULE RUL_LowerCase
AS
@ruleval >='a' AND @ruleval <='z'

'la associo alla colonna department

sp_bindrule RUL_LowerCase,'Employees.department'


'Per Mostrare le Rules

sp_help Nome_Rule;
```

Vedere il contenuto delle Rules

sp_helptext nome_rule

```
sp_helptext department_values

Text
----------------------------------------------------------------------------------
create rule department_values
as @department in ('Sales','Field Service','Logistics','Software')
```

Disassiociare Le Rules

sp_unbindrule table_name.column or user_datatype [, futureonly]

```
'Vedere la struttura della tabella Employees
sp_help employees;

'Elimino il bound
sp_unbindrule 'employees.department'
```

Rinominare le Rules

sp_rename object_name, new_name

Eliminare le Rule

DROP RULE rule_name_1[,...rule_name_n]

'Prima si elimina l'associazione alle tabelle poi si elimina la regola

```
drop rule department_values
```

Le Procedure memorizzate

Una procedura di sistema è una procedura memorizzata che SQLServer ha creato e incluso per svolgere compiti di carattere amministrativo ed informativo. In sostanza queste procedure memorizzate rappresentano blocchi di codice memorizzati nel sistema le quali permettono di effettuare certe operazioni lato server. Si può, cioè, chiamare la procedura assegnandogli i 'parametri' necessari per il suo funzionamento e questa eseguirà il codice che la compone. La cosa piacevole è che l'elaborazione della procedura avviene sul server in cui SQLServer risiede.
In questo modo si riduce il traffico di rete in quanto da remoto ho l'invio di una richiesta (eseguire laprocedura X) assegnando i parametri richiesti per l'elaborazione. Il server elabora la procedura e restituisce la risposta. Come si vede in rete viaggerà solo la domanda e la risposta.
In SQLServer esistono tre tipi di procedure:
Le procedure, le function e i package.

Procedure di sistema
Procedure estese
Proceduredefinite dall'utente

Procedure di Sistema

Una procedura di sistema è una procedura memorizzata che è stata inclusa nel sistema per svolgere compiti di carattere amministrativo ed informativo.Le procedure di sistema sono precedute dal prefisso sp seguito da un carattere di tottolineatura .*Quindi,* ad esempio, la procedura *sphelp* fornisce informazioni ed è realizzata da SQLServer.

Categoria	Descrizione	Elenco
Catalog Procedures	Procedure per avere informazioni sull'ambiente in cui si lavora.	sp_column_privileges sp_columns sp_sproc_columns sp_databases sp_statistics sp_fkeys sp_stored_procedures sp_pkeys sp_table_privileges sp_server_info sp_tables
Cursor Procedures	Implementano le funzionalità dei cursori	sp_cursor_list sp_describe_cursor_tables sp_describe_cursor sp_describe_cursor_columns
Distributed Queries Procedures	Usate per implementare e gestire Query distribuite	sp_addlinkedserver sp_indexes sp_addlinkedsrvlogin sp_linkedservers sp_catalogs sp_primarykeys sp_column_privileges_ex sp_columns_ex sp_table_privileges_ex sp_droplinkedsrvlogin sp_tables_ex sp_foreignkeys
Security Procedures	Usate per gestire la sicurezza	sp_addalias sp_droprole sp_addapprole

		sp_droprolemember sp_addgroup sp_dropserver sp_addlinkedsrvlogin sp_dropsrvrolemember sp_addlogin sp_dropuser sp_addremotelogin sp_grantdbaccess sp_addrole sp_grantlogin sp_addrolemember sp_helpdbfixedrole sp_addserver sp_helpgroup sp_addsrvrolemember sp_helplinkedsrvlogin sp_adduser sp_helplogins sp_approlepassword sp_helpntgroup sp_change_users_login sp_changedbowner sp_helprole sp_changegroup sp_helprolemember sp_changeobjectowner sp_dbfixedrolepermission sp_helpsrvrole sp_defaultdb sp_helpsrvrolemember sp_defaultlanguage sp_helpuser sp_denylogin sp_password sp_dropalias sp_remoteoption sp_dropapprole sp_revokedbaccess sp_dropgroup sp_revokelogin sp_droplinkedsrvlogin sp_setapprole sp_droplogin sp_srvrolepermission sp_dropremotelogin sp_validatelogins
System Procedures	Usate per il generale mantenimento di SQLServer	sp_add_data_file_recover_suspect_db sp_helpconstraint sp_add_log_file_recover_suspect_db sp_helpdb sp_addextendedproc sp_helpdevice sp_addmessage sp_helpextendedproc sp_addtype sp_helpfile sp_addumpdevice

		sp_helpfilegroup sp_altermessage sp_help_fulltext_catalogs sp_autostats sp_help_fulltext_catalogs_cursor sp_attach_db sp_help_fulltext_columns sp_attach_single_file_db sp_help_fulltext_columns_cursor sp_bindefault sp_help_fulltext_tables sp_bindrule sp_help_fulltext_tables_cursor sp_bindsession sp_helpindex sp_certify_removable sp_helplanguage sp_configure sp_helpserver sp_create_removable sp_helpsort sp_createstats sp_helptext sp_cycle_errorlog sp_helptrigger sp_datatype_info sp_indexoption sp_dbcmptlevel sp_lock sp_dboption sp_monitor sp_delete_backuphistory sp_processmail sp_depends sp_procoption sp_detach_db sp_recompile sp_dropdevice sp_refreshview sp_dropextendedproc sp_rename sp_dropmessage sp_renamedb sp_droptype sp_serveroption sp_executesql sp_setnetname sp_getbindtoken sp_spaceused sp_fulltext_catalog sp_tableoption sp_fulltext_column sp_unbindefault sp_fulltext_database sp_unbindrule sp_fulltext_service sp_updatestats sp_fulltext_table sp_validname

		sp_help sp_who
Altre categorie	Vedi Help on line	

Uso di alcune procedure di sistema

sp_databases

Restituisce l'elenco dei Database presenti su un Host

sp_columns

Restituisce le colonne di una tabella

```
sp_columns [@table_name =] object
        [,[@table_owner =] owner]
        [,[@table_qualifier =] qualifier]
        [,[@column_name =] column]
        [,[@ODBCVer =] ODBCVer]
```

```
sp_columns employees
```

sp_tables

Restituisce le tabelle di un DB

```
sp_tables [[@name =] 'name']
        [,[@owner =] 'owner']
        [,[@qualifier =] 'qualifier']
        [,[@type =] "type"]
```

```
name=Nomedb
```

sp_stored_procedures

```
sp_stored_procedures [[@sp_name =] 'name']
        [,[@sp_owner =] 'owner']
        [,[@sp_qualifier =] 'qualifier']
```

```
name=Nomedb
```

sp_help

```
sp_help [[@objname =] name]
```

sp_helptext

```
sp_helptext [@objname =] 'name'
```

Procedure estese

Una procedura estesa è simile ad una procedura di sistema o a un a procedura definita dall'utente,tranne per il fatto che la procedura non è memorizzata in SQL Server,ma in una DLL

oppure in qualche altro file o processo esterno.Un esempio di procedura esterna è un'interfaccia con SQLServer dal proprio sistema di posta elettronica..In base ad una convenzione una procedura estesa di solito inizia con il prefisso **xp** seguito dal carattere di sottolineatura.
SQLServer si appoggia per queste funzioni a delle librerie presenti nel sistema (DDL sta per Dynamically Linked Library).All'interno di qwueste librerie ci sono delle funzioni che le procedure memorizzate estese possono chiamareper ottenere un determinato risultato.Ad esempio si vuol scollegare qualcuno dalla rete,funzione che non è presente nulle funzioni standard di SQLServer.
In sostanza le procedure estese permettono di accedere alle funzioni di basso livello che sono presenti nel sistema operativo oppure sono create appositamente, ad esempio con un Visual Basic.

Creare Procedure memorizzate

Le procedure memorizzate create dall'utente è uno strumento potente a disposizione in un qualsiasi RDBMS server.Nelle applicazioni client/server si possono richiamare delle procedure residenti sul server che mi fanno certe operazioni restituendomi un dato risultato.Questo fa si che l'elaborazione risieda principalmente sul server facendo correre in rete solo domanda e risposta.Le procedure memorizzate sono costituite da codice SQL che usa istruzioni di controllo (IF,THEN ELSE),strutture iterative e variabili.

Strutture di controllo

Le strutture di controllo consentono di intervenire sul flusso di esecuzione del programma. In assenza delle istruzioni per il controllo del flusso, il programma seguirebbe una logica unidirezionale, eseguendo le istruzioni da sinistra a destra e dall'alto al basso. Alcuni programmi molto semplici possono essere scritti seguendo tale logica unidirezionale. Nonostante alcuni flussi possano essere gestiti utilizzando operatori che consentono di stabilire la precedenza di alcune operazioni rispetto ad altre, la validità e l'utilità di un linguaggio di programmazione dipendono comunque dalla possibilità di modificare l'ordine delle istruzioni utilizzando strutture e cicli.

Strutture decisionali

I blocchi di codice T-SQL possono verificare condizioni specifiche e quindi eseguire le diverse operazioni in base al risultato di tale verifica. T-SQL supporta le seguenti strutture decisionali:

- If...Else
- Begin...End
- Select Case

If...Else

La struttura If...Else consente di eseguire una o più istruzioni in base alle condizioni specificate. La sintassi può essere a riga singola o a blocco, ovvero a più righe:

```
IF expression
        statement
[ELSE]
       [IF expression]
           statement]
```

La expression è in genere un confronto, ma può essere anche un'espressione che restituisce un valore numerico interpretato automaticamente come True o False. I valori numerici uguali a zero sono considerati False, mentre quelli diversi da zero sono considerati True. Se la *condizione* risulta True, vengono eseguite tutte le statement successive altrimenti il blocco dopo ELSE.

'Vediamo un esempio.Cerco il record che ha codice 'Frank'.Se questo esiste si scriverà record 'esistente nella parte inferiore dello schermo.

```
if exists (select * from Customers
where CustomerID='Frank')
    print 'record Esistente'
```

‘Posso inserire una alternativa nel caso in cui il record sia non esistente attraverso ELSE.

```
if exists (select * from Customers
where CustomerID='Frank________')
   print 'record Esistente'
else
print 'nessun record'
```

Begin...End

Si usano le parole chiave Begin…End per indicare un set di istruzioni di T-SQL che devono essere eseguite unite come **un unico blocco di istruzioni**.Spesso si usa Begin…End all’interno di strutture condizionali.Viene usato per permettere l’uso di più istruzioni T-SQL all’interno della stessa struttura condizionale.

```
BEGIN
    statements
END
```

```
if exists (select * from employees
where EmployeeID=5)
     begin
          print 'Record Presente'
          select Firstname,LastName from employees
          where EmployeeID=5
     end
```

‘Se si aggiunge la parola chiave ELSE

```
if exists (select * from employees
where department='Logistics')
     begin
          print 'Record Presente'
          select Firstname,LastName from employees
          where department='Logistics'
     end
else print 'Nessuna Riga'
```

Select Case

Si usa l'istruzione CASE quando si devono prendere decisioni basate su molteplici opzioni.

```
CASE [expression]
WHEN simple expression1|Boolean expression1 THEN expression1
[[WHEN simple expression2|Boolean expression2
   THEN expression2] [...]]
    [ELSE expressionN]
END
```

```
select Lastname,Firstname,division=
case department
     when "SOFTWARE" then "Realizzazione Software"
     when "Logistics" then "Logistica/Supporto"
     when "Admin" then "Amministrazione"
     else "Other department"
end,
EmployeeID
from Employees
```

```
Lastname             Firstname  division               EmployeeID
-------------------- ---------- ---------------------- -----------
Davolio              Nancy      Other department       1
Fuller               Andrew     Other department       2
Leverling            Janet      Other department       3
Peacock              Margaret   Other department       4
Buchanan             Steven     Other department       5
Suyama               Michael    Other department       6
King                 Robert     Other department       7
Callahan             Laura      Other department       8
Dodsworth            Anne       Other department       9
Rossi                Mario      Logistica/Supporto     13
Verdi                Giuseppe   Realizzazione Software 14
Bianchi              Giulio     Realizzazione Software 15
Gori                 Franca     Realizzazione Software 16
Ferrini              Alberta    Logistica/Supporto     19
```

Strutture iterative

Le strutture iterative consentono di eseguire ripetutamente una o più righe di codice T-SQL.T-

While

While è una parola chiave definisce una condizione che esegue una istruzione T-SQL 1 o n volte fino a che questa condizione è verificata.

```
WHILE
      <boolean_expression>
      <sql_statement>
```

'Nell'esempio seguiente While esegui l'istruzione select per un numero <10 volte scrivendo una riga 'ed incrementando una variabile il cui valore determina il loop.

```
declare @x int
select @x=1
while @x<10
begin
```

```
print 'Ripeti meno di 10 volte'
select @x=@x+1
end
```

‘Notate che è stata definita una variabile unando l’istruzione DECLARE. Sempre una variabile è ‘preceduta dal simbolo @ come un parametro.

Si una **Break** per uscire dal loop

```
declare @x int
select @x=1
while @x<5
begin
    print 'Quando conto 3 esco.Due righe verranno scritte'
    select @x=@x+1
        if @x=3
        break
end
```

Ogni istruzione dopo **Continue** viene ignorata.A volte viene attivata con una istruzione If

```
declare @x int
select @x=1
while @x<5
begin
    print 'Saranno quattro righe'
    select @x=@x+1

continue
    print 'Questa riga di codice sarà ignorata'
end
```

Altro esempio:

```
declare @x int
declare @y tinyint
select @x=1, @y=1
while @x<5
begin
    print 'Sarà una sola riga'
    select @x=@x+1
    select @y=@y+1
    if @y=2
    begin
```

```
        print 'Se y è = 2 si attiva il Break'
        break
    end
    end
print 'Sono Fuori dal while loop'
```

Definizione ed uso delle variabili

DECLARE @variable_name datatype [,variable_name datatype...]

La dichiarazione delle variabili viene effettuata con l'istruzione Descare.
Il nome della variabile deve essere preceduto dal simbolo@
Datatype è il tipo di dato.

L'assegnazione dei valori avviene con l'istruzione select

```
SELECT @variable_name = expression |select statement
[,@variable_name = expression select statement]
[FROM list of tables] [WHERE expression]
[GROUP BY...
[HAVING ...]
[ORDER BY...]
```

Il valore della variabile può essere una espressione oppure il risultato di una select.

'In questo esempio si dichiara una variabile @mynum come tipo di dato int
'Si assegna il valore del numero righe presenti nella tabella employees
'Si converte il valore numerico in carattere assegnandolo alla variabile @mychar
'Si dischiara la variabile @mess come char(40) e si assegna la stringa= 'Ci sono ' + @mychar + '
'righe nella tabella Employees'
'Si stampa il contenuto della variabile
'E' necessario convertire la variabile numerica in stringa per poterla congiungere con il resto e versarla
'nella variabile @mess.

```
declare @mynum int
select @mynum = count(*)from Employees
declare @mychar char(2)
select @mychar = convert(char(2),@mynum)
declare @mess char(40)
select @mess ='Ci sono ' + @mychar + ' righe nella tabella Employees'
print @mess
```

Le variabili globali

Una variabile globale viene definita da SQLServer.Non si possono definire variabili globali con le routines.Si possono usare le variabili globali predefinite.Si fa riferimento alle variabili globali con il doppio simbolo @@.Ovviamente non si possono definire variabili con gli stessi nomi delle variabili globali.

Global Variables for Microsoft SQL Server

	Description

Global Variable	
@@CONNECTIONS	*total logons or attempted logins*
@@CPU_BUSY	*cumulative CPU Server time in ticks*
@@DBTS	*value of unique timestamp for database*
@@ERROR	*last system error number : 0 if successful*
@@FETCH_STATUS	*status of the last FETCH statement*
@@IDENTITY	*the last inserted identity value*
@@IDLE	*cumulative CPU Server idle time*
@@IO_BUSY	*cumulative Server I/O time*
@@LANGID	*current language ID*
@@LANGUAGE	*current language name*
@@MAX_CONNECTIONS	*max simultaneous connections*
@@MAX_PRECISION	*precision level for decimal and numeric datatypes.*
@@MICROSOFTVERSION	*internal version number of SQL Server*
@@NESTLEVEL	*current nested level of calling routines from 0 to 16*
@@PACK_RECEIVED	*number of input packets read*
@@PACKET_SENT	*number of output packets written*
@@PACKET_ERRORS	*number of read and write packet errors*
@@PROCID	*current stored procedure ID*
@@ROWCOUNT	*number of rows affected by last query*
@@SERVERNAME	*name of local server*
@@SERVICENAME	*name of the running service*
@@SPID	*current process server ID*
@@TEXTSIZE	*current of max text or image data with default of 4K*
@@TIMETICKS	*number of microseconds per tick-machine independent. tick is 31.25 milliseconds/1/32 sec.*
@@TOTAL_ERRORS	*number of errors during reads or writes*
@@TOTAL_READ	*number of disk reads (not cache)*
@@TOTAL_WRITE	*number of disk writes*
@@TRANCOUNT	*current user total active transactions*
@@VERSION	*date and version of SQL Server*

In questo esempio viene usata la variabile globale *@@SERVERNAME* per ritrovare la versione corrente di SQLServer

```
PRINT @@VERSION
declare @mess1 char(21)
select @mess1 = 'Server name is ' + @@servername
PRINT @mess1
```

Le Variabili

Nelle Stored procedures possono essere dichiarate delle variabili.Le variabili anno un ambito di validità che si riferisce alle sole stored procedures.Il concetto è quello delle sub e function di Visual Basic o di qualsiasi altro linguaggio di programmazione guidato da eventi. Ogni variabile è preceduta dal simbolo @ e la dichiarazione segue la seguente sintassi:

```
DECLARE @variable_name datatype [,variable_name datatype...]
```

Alla variabile possono essere assegnati valori che provengono anche da query.

Procedure aggiuntive e Parole chiave 'Bach'

Queste parole chiave non possono essere inserite in una specifica categoria ma possono essere assimilate alle funzioni.
Alcune di queste parole sono: GOTO,RETURN, WAITFOR.

GOTO

Devia il codice alla label stabilita

GOTO label

```
declare @count smallint
select @count =1
restart:
print 'si'
select @count =@count + 1
while @count <= 4
goto restart
```

Return

Permette di uscire da una routine o qyery con la possibilità di attrinuire alla routine un valore.Si può usare Return in ogni punto della procedura.Ogni riga che segue RETURN non viene esegiuta.Quindi,è come Break con la differenza che return restituisce un valore.

RETURN [integer]

Integer=E' un valore di ritorno.

I valori seguinti mostrano all'uscita di una routine il risultato dell'esecuzione della routine

Selected Microsoft SQL Server Status Values

Return Value	*Meaning*
0	*successful execution*
–1	*missing object*
–2	*datatype error*
–3	*process was chosen as a deadlock victim*
–4	*permission error*
–5	*syntax error*
–6	*miscellaneous user error*
–7	*resource error, such as out of space*
–8	*nonfatal internal problem*
–9	*system limit was reached*
–10	*fatal internal inconsistency*
–11	*fatal internal inconsistency*
–12	*table or index is corrupt*
–13	*database is corrupt*
–14	*hardware error*

Un altro modo di utilizzare Return è quello ad esempio di attribuirgli all'interno di una routine un valore richiamandolo ed attribuendolo ad una variabile con la seguente sintassi.

EXEC[ute] @return_status=procedure_name

A. Uscita da una procedura

```
CREATE PROCEDURE findjobs @nm sysname = NULL

AS
IF @nm IS NULL
BEGIN
        PRINT 'You must give a username'
        RETURN
    END
ELSE
    BEGIN
        SELECT o.name, o.id, o.uid
        FROM sysobjects o INNER JOIN master..syslogins l
            ON o.uid = l.suid
        WHERE l.name = @nm
    END
```

In questo caso l'uscita determina o un valore positivo o negativo nel caso di errore.

B. Richiamo valore

```
CREATE PROCEDURE checkCitta @param varchar(11)

AS

IF (SELECT City FROM Employees WHERE EmployeeID = @param) = 'London'

    RETURN 1
ELSE
    RETURN 2
```

'restituisce il valore

```
DECLARE @return_city int
EXEC @return_city = checkCitta '6'
SELECT 'Città Restituita' = @return_city
```

```
Città Restituita
----------------
1

(1 row(s) affected)
```

Waitfor

Indica un tempo o un intervallo di tempo o un evento che attiva l'esecuzione di un blocco, di una procedura o di una transazione

WAITFOR {DELAY 'time' | TIME 'time'}

DELAY=indica di aspettare l'esecuzione un certo lasso di tempo.Max 24 ore
TIME=Scatta l'esecuzione ad un'ora specifica

```
waitfor delay '00:00:40'
select * from employees

waitfor time '15:10:51'
select * from employees
```

Le procedure Registrate definite dall'utente

Le procedure registrare definite dall'utente permettono l'astrazione del linguaggio SQL.

I vantaggi stanno principalmente nella riduzione del traffico di rete, nella possibilità di utilizzare istruzioni di controllo dei flussi(IF …ELSE …WHEN …ELSE…END),miglioramento dell'esecuzione del codice, nel codificare l'utilizzo di una base di dati con un piano di procedure prestabilite.

Sintassi

```
CREATE PROC[EDURE] [owner,] procedure_name [;number]
[@parameter_name datatype [=default_value] [OUTput]
...
[@parameter_name datatype [=default_value] [OUTput]
 [WITH {RECOMPILE|ENCRYPTION}]|[FOR REPLICATION]
AS sql_statements
```

- **procedure_name**=Nome della procedura
- **number**=permette di raggruppare le procedure
- **@parameter_name**=parametro che può essere usato nella procedura
- **datatype**=tipo di dato del parametro
- **default_value**=valore di default per il parametro
- **OUTput**=-Indica che il nome del parametro viene usato per l'output.
- **WITH RECOMPILE**=queste due parole chiave opzionali dicono a SQLServer di crittografare il testodella procedura memorizzata nella taqbella di sistema *syscomments*
- **FOR REPLICATION**= segnala che la procedura memorizzata verrà utilizzata per replica,cioè SQLServer deve ricompilare l'istruzione prima diutilizzarla.Non si possono usare contemporaneamente le istruzioni **FOR REPLICATION** e **WITH RECOMPILE**
- **sql_statements** = è una qualsiasi istruzione SQL valida per la quale si hanno le necessarie autorizzazioni.

Se si tenta di modificare una stored procedure e si salva la versione modificata nello stesso database SQLServer 6.5 e versioni precedenti,viene inviato un messaggio di errore.Di conseguenza, è consigliabile anteporre alle istruzioni CREATE PROC[EDURE] il seguente blocco di istruzioni:

```
IF EXISTS (SELECT * FROM sysobjects
        WHERE id=object_id ('dbo.proc_name')
        AND sysstat & 0xf =4)
DROP PROCEDURE  dbo.proc_name
```

Siccome SQLServer 7 supporta l'istruzione ALTER PROC[EDURE] è possibile sostituire PROC[EDURE] con ALTER senza il blocco sopra.

Esempio:

```
IF EXISTS (SELECT * FROM sysobjects
        WHERE name='ddg_OrdersByCountry' AND type='P')
DROP PROCEDURE  ddg_OrdersByCountry
GO
USE Northwind
GO
CREATE PROC ddg_OrdersByCountry
        @ShipCountry varchar (15)='USA'
AS SELECT * FROM Orders WHERE ShipCountry = @ShipCountry
```

Se si usa il database Northwind non è necassaria l'istruzione USE che determina il passaggio da un database all'altro in SQLServer.

Mentre la procedura scritta in precedenza è valida per versioni < di 6.5 la sguente è valida per SQLServer 7 e successive.

```
ALTER PROC ddg_OrdersByCountry
        @ShipCountry varchar (15)='USA'
AS SELECT * FROM Orders WHERE ShipCountry = @ShipCountry
```

Per eseguire le procedure è necessario usare la parola riservata EXEC[UTE] seguita dal nome di routine e da un elenco separato da virgole di valori di parametro input,se possibile.

```
EXEC ddg_OrdersByCountry
```

In questo caso il parametro utilizza il default =’USA’. Qualora voglia utilizzare ‘Germany’ come valore di parametro dovrò scrivere in questo modo:

```
EXEC ddg_OrdersByCountry 'germany'
```

Convenzioni di denominazione

LA denominazione di una stored procedure varia a seconda dell’utilizzo che si intende fare della stessa.Si propongono le seguenti denominazioni:

- Utilizzare un prefisso da 2 a 4 caratteri più + la sottolineatura per identificare il database o le applicazioni che usano il database.Es nw_ nwin_ indicano che si usa il database northwind.
- Accodare il nome della tabella coinvolta nella procedura eventualmente utilizzando delle abbreviazioni. nw_cust (cust sta per customer).
- Accodare l’abbreviazione dell’operazione che si va ad effettuare con la stored procedure(ins,del,upd) oppure il nome dell’oggetto di destinazione di una stored(es. se il set di dati andrà a riempire una List useremo l’abbreviazione lst.Quindi avremo ad esempio nw_custupd,nw_custLst oppure nw_custData.
- Se la storede richiede l’impiego della clausola where accudare un simbolo di sottolineatura e il nome del campo del vincolo. Se, ad esempio, il vincolo è CustomerID la stored diverrà nw_custlst_CustID
- Se la stored richiede una funzione idAggregazione allora bisogna accodare un simbolo di sottolineatura + il nome del campo oggetto della funzione+ la funzione.Esempio: nw_cust_CustIDMax

Quando e perché creare le Stored Procedures

Abbiamo già discusso dei vantaqggi delle procedure registrate.E’ una delle grandi peculiarità di un RDBMS.Cosa offre una procedura registrata:

1) Riduzione del traffico di rete
2) Utilizzo dei parametri
3) Utilizzo delle strutture condizionali e cicliche.

Dal punto di vista della costruzione una Stored procedure è assimilabile ad una query con parametri di Microsoft Access, ma si differenzia da questa per la possibilità di impiegare nel codice strutture condizionali e cicliche,cosa che non è permessa con le parametriche.Inoltre,solo le stored procedures realizzano una elaborazione su server al contrario delle parametriche. La grossa difficoltà nell’operare con le Stored non è tanto la loro realizzazione ma è la progettazione di un complesso di procedure adeguiate agli scopi del nostro db o della nostra applicazione.Le stored sono statiche e quindi difficilmente sostituibili o modificabili all’interno dell’economia di una applicazione che fa base su esse.Sbagliare il ‘piano di realizzazione’ deelle stored può voler dire reingegnerizzare totalmenti il software o i software che su esse si basano. Vuol dire rimettere mano al db Server con grosso spreco di risorse umane.

A. CREAZIONE DI STORED PROCEDURES CON QUERY ANALYSER

- Creo una procedure che aggiunge una regione alla tabella region di Northwind
- Eseguo la query in Query Analyser

La tabella region contiene 2 campi:Il campo regionID che è un intero.A sua volta è chiave primaria e quindi necessario e non duplicabile.Occorre per questo assegnargli un valore nuovo e che non sia presente nelle righe precedenti. Per questo si dichiara una variabile @contatore che seleziona il valore piu elevato di RegionID e lo si incrementa. Si crea un vero e proprio campo contatore. Successivamente si inserisce una istruzione Insert INTO nella quale immettiamo il valore delle due variabili @contatore,ricavato dall'incremento del valore max della RegionID, e @nuova_regione. Controllare con i tasti CTRL+F5 la sintassi della Stored.

```
CREATE PROC nw_regione @nuova_regione nchar (50) as
DECLARE @contatore int
select @contatore= max(RegionID) from region

SELECT @contatore=@contatore +1

INSERT INTO REGION
VALUES(@contatore,@nuova_regione);
```

'inserisco la una riga

```
EXEC nw_regione 'Toscana';
```

B. CREAZIONE DI STORED PROCEDURES CON ENTERPRISE MANAGER

Avviare SQL Server Enterprise Manager e scegliere il db Northwind.Espandere l'albero del db Northwind e selezionare stored procedures.Scegliere new procedure.Si apre a questo punto una finestra che permette di editare la stored e di controllare la sintassi.La scheda permissiones permette di attribuire le autorizzazioni per gruppi di utenti.

Utilizzo di Visual Data Tools a delle stored procedure

Una alternativa molto utile per la creazione di stored procedure è l'utilizzo di VDT(Visual Data Tools). Avviare un nuovo progetto VB e assegnare un db SQL Server (Northwind) come prima connessione).Si Aprirà una finestra detta Data view con un albero simile a quello di SQL Server ma relativo alla sola connessione in atto (cioè northwind).A questo punto scegliere la cartella stored procedure e scegliere di modificare o di creare nuove stored con il pulsante destro del mouse.

Codice Visual Basic per attivazione di una Stored procedure

```
Sub stProcedure_Click()
Dim cnn1 As ADODB.Connection

Dim cmdUomoDonna As ADODB.Command
Dim prmUomoDonna As ADODB.Parameter
Dim rstUomoDonna As ADODB.Recordset
Dim strMsMR As String
Dim strCnn As String
Dim numRec As Long
Dim row As Long


'definizione della connessione (stringa di connessione)
Set cnn1 = New ADODB.Connection
strCnn = "Provider=SQLOLEDB.1;Integrated Security=SSPI;Persist Security Info=False;Initial
Catalog=Northwind;Data Source=MAURI"
cnn1.Open strCnn

' Definire un oggetto command per una stored procedure.
```

```
Set cmdUomoDonna = New ADODB.Command
Set cmdUomoDonna.ActiveConnection = cnn1

' Definire il parametro per la stored procedure.

strMsMR = Trim(InputBox("Uomo MR. o Donna MS."))

'definizione della Stored da attivare

cmdUomoDonna.CommandText = "UomoDonna ('" & strMsMR & "')"
cmdUomoDonna.CommandType = adCmdStoredProc
Set rstUomoDonna = cmdUomoDonna.Execute


Do
Debug.Print rstUomoDonna!lastName

rstUomoDonna.MoveNext
Loop Until rstUomoDonna.EOF

rstUomoDonna.Close
cnn1.Close


End Sub

‘-------------------------------------------------UTILIZZO DI ASP/SQLServer

<%@LANGUAGE="VBSCRIPT"%>

 <%

set mbkNazione = Server.CreateObject("ADODB.Command")
mbkNazione.ActiveConnection = "dsn=MBOOK;uid=sa;"

mbkNazione.CommandText = "dbo.mbkNazione ('b')"
mbkNazione.CommandType = 4
mbkNazione.CommandTimeout = 0
rsNazione = mbkNazione.Execute
%>
<html>
<head>
<title>Prova di collegamento ASP/SQLServer</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
</head>

<body bgcolor="#FFFFFF">
<%= rsNazione("NAZIONE").Value %>
</body>
</html>
```

I Cursori

Un cursore può essere definito come un sistema per manipolare un *gruppo di risultati*. Un gruppo di risultati è un insieme di dati che corrisponde alle richieste dell’interrogazione.Può contenere qualsiasi numero di righe di dati. Un cursore è un sottoinsieme definibile dall’utente dell’intero gruppo di risultati e può essere costituito da 1 o più righe. Utilizzare un cursore permette di lavorare con un set di dati inferiore ma genera ovviamente un overhead dovuto al compimento di una istruzione in più per generare il set di risultati..
In SQL Server un cursore deve essere dichiarato.
Esistono 4 tipi di cursori:

Statico. Occupa poche risorse ma è in sola lettura e non si accorge dei dati cambiati.
Dinamico. Richiede più risorse ma è R/W e si accorge dei cambiamenti dei dati.
Solo Avanti.Si accorge dei cambiamenti dei nelle righe che non sono state ancora recuperate.Non può scorrere indietro.
Pilotato da Keyset. I cambiamenti alle variabili Keyset sono permessi tramite variabile. Non sdi accorge dei cambiamenti nelle colonne non Keyset.

Dichiarare il cursore

Per poter essere usati i cursori vanno prima dichiarati.

```
DECLARE name_of_cursor [INSENSITIVE] [SCROLL] CURSOR
FOR Select_Statement
[FOR {READ ONLY | UPDATE [OF Column_List]}]
```

INSENSITIVE=se presente indica che i valori sono stati registrati in una tabella temporanea del tempdb
SCROLL=se presente indica che il cursore può muoversi in tutte le direzioni
FOR READ ONLY=se è presente indica che è un cursore di sola lettura
FOR UPDATE [OF *Column_List*] lista opzionale di colonne che possono essere aggiornate se specificato con for update

```
Declare Cur_Empl Cursor
For   Select EMPLOYEEID,LASTNAME
    From  EMPLOYEES
    Order By EMPLOYEEID
Go
```

Aprire il cursore

All'apertura del cursore viene generato un gruppo di record.Quando il cursore è aperto, tale gruppo di record definito in DECLARE del cursore viene popolato

```
OPEN nomecursore
```

```
open Cur_Empl
```

Usare il cursore

Dopo che il cursore è stato dichiarato e aperto possiamo usarlo.Si utilizza la parola chiave FETCH.Ricordiamo che si tratta di un set di dati.

```
FETCH [[NEXT | PRIOR | FIRST | LAST |
       ABSOLUTE n/@nvar | RELATIVE n/@nvar ]
FROM] cursor_name
[INTO @variable_name1, @variable_name2]
```

NEXT= viene recuperata la riga immediatamente successiva a quella corrente nel cursore
PRIOR= viene recuperata la riga immediatamente prima a quela corrente nel cursore
FIRST= viene recuperata la prima riga del cursore
LAST= viene recuperata l'ultima riga del cursore
ABSOLUTE=Sposta la riga corrente del cursore di n righe in avanti all'inizio del cursore (se n è negativo dalla fine del cursore).
RELATIVE=Sposta la riga corrente del cursore di n righe in avanti rispetto alla posizione corrente (se n è negativo indietro).
INTO= è il nome della variabile destinata a contenere il gruppo di risultati FETCH.Se si usa INTO si devono specificare abbastanza variabili per contenere tutte le colonne restituite dall'operazione di recupero.

‘attenzione il cursore è sono in avanti
fetch next from Cur_Empl

‘si luppa il fetch e si inserisce il valore in una variabile @@

```
while @@Fetch_status=0
begin
        fetch next
        from Cur_Empl
end
```

Chiudere il cursore

Se si chiude il cursore si eliminano i record dall’area di memoria riservata al cursore.

CLOSE nomecursore

CLOSE Cur_Empl

Eliminare il cursore

Eliminare il cursore vuol dire eliminare ‘deallocare’ l’area di memoria riservata. Si usa la parola chiave DEALLOCATE. Qualsiasi variabile dichiarata con DECLARE assegna uno spazio di memoria.Questo spazio viene eliminato con DEALLOCATE.

DEALLOCATE CUR_Empl

Gli Attivatori (Triggers)

Gli attivatori costituiscono un sistema importante per migliorare l’integrità referenziale.Un attivatore entra in funziuone quando si verificano una serie di eventi definiti dal programmatore negli attivatori stessi.Gli Attivatori si usano per gestire meglio operazioni come INSERT,UPDATE e DELETE.

```
CREATE TRIGGER [owner.]trigger_name
ON [owner.]table_name
FOR {INSERT, UPDATE, DELETE}
AS sql_statements
```

Esempio

1.Crea Tabella ContaRighe

```
CREATE TABLE ContaRighe(
Impiegato varchar (10)
)
```

2.Crea il Trigger. Questo Trigger inserisce una riga nella tabella ContaRighe tutte le volte cheinserisco un nuovo record nella tabella LAVORATORE

```
CREATE TRIGGER TRG_insert_Lav2
On LAVORATORE
FOR INSERT
AS
Insert ContaRighe(IMPIEGATO)
select CodLav
FROM inserted
```

3.Inserisci Riga

```
INSERT INTO LAVORATORE VALUES ('098564','00780','GIANNI','MORANDI','QF06','201','POLO6
BIBL.DI MEDICINA','V.LE MORGAGNI, 85','FIRENZE','50100','FI',null,null,null);
```

4. Eleimina Trigger

```
DROP TRIGGER TRG_insert_Lav2;
```

Con SP_HELP si vede tutte le caratteristiche della tabella

```
sp_help LAVORATORE;
```

Vedo il testo del Trigger

```
sp_helptext TRG_insert_Lav2;
```

Per vedere meglio l'elenco dei Triggers della tabella usare Enterprise manager.Puntarsi sulla tasbella e premere il tasto destro del mouse.Selezionare All Tascks

Transazioni (DCL)

La gestione delle transazioni è una aspetto molto importante nei database. Quando la creazione, la modifica e la cancellazione dei dati impegna più righe in tabelle diverse può capitare che per qualche motivo non vada tutto a buon fine. Si possono creare, quindi, delle situazioni molto pericolose in cui alcuni dati possono essere aggiornati in alcune tabelle altri no. Le transazioni rappresentano un blocco univoco di molteplici operazioni che si possono realizzare su 1 come su n tabelle.
La transazione è un blocco unico composto di n operazioni per cui se tutte le operazioni sono andate a buon fine il database si aggiorna altrimenti rimane nello stato antecedente la transazione.
I comandi BEGIN TRANSACTION, COMMIT TRANSACTION,ROLLBACK TRANSACTION controllano l'esito delle transazioni.

BEGIN TRANSACTION Indica l'inizio del gruppo di operazioni che devono essere considerate come una unica entità (Transazione)
COMMIT TRANSACTION serve per confermare una transazione rendendo i cambiamenti prodotti irreversibili.
ROLLBACK TRANSACTION serve per annullare le modifiche

ESEMPIO:

```
BEGIN TRANSACTION

INSERT INTO LAVORATORE VALUES ('078888','00780','EROS','RAMAZZOTTI','QF06','201','POLO6
BIBL.DI MEDICINA','V.LE CANTERINO, 85','ROMA','50100','FI',null,null,null);

INSERT INTO LAVORATORE VALUES
('000005','00780','LUCIANO','PAVAROTTI','QF06','201','POLO6 BIBL.DI MEDICINA','V.LE
MORGAGNI, 85','PARMA','50100','FI',null,null,null);

INSERT INTO LAVORATORE VALUES ('000007','00780','MARIA','CALLAS','QF06','201','POLO6
BIBL.DI MEDICINA','V.LE MORGAGNI, 85','FIRENZE','50100','FI',null,null,null);

IF @@Error=0
        COMMIT TRANSACTION
ELSE
        ROLLBACK TRANSACTION
```