



paolo  
atzeni

stefano  
ceri

piero  
fraternali

stefano  
paraboschi

riccardo  
torlone

IV edizione

# basi di dati

appendici applicative

---

**Mc  
Graw  
Hill**  
Education

Paolo Atzeni  
Stefano Ceri  
Piero Fraternali  
Stefano Paraboschi  
Riccardo Torlone

# **Basi di dati**

## **Appendici applicative**

### **Quarta edizione**

**McGraw-Hill**

---

**Milano** • New York • San Francisco • Washington D.C. • Auckland  
Bogotá • Lisboa • London • Madrid • Mexico City • Montreal  
New Delhi • San Juan • Singapore • Sydney • Toronto

Copyright ©  
McGraw-Hill Education (Italy), s.r.l.  
Via Ripamonti, 89  
20141 Milano



Le Appendici contenute in questo ebook sono state realizzate come materiale integrativo per i due seguenti volumi:

- P. Atzeni, S. Ceri, P. Fraternali, S. Paraboschi, R. Torlone, Basi di dati. Modelli e linguaggi di interrogazione, 4e, McGraw-Hill Education (Italy) srl, ISBN 978-88-386-6800-5
- P. Atzeni, S. Ceri, P. Fraternali, S. Paraboschi, R. Torlone, Basi di dati, 4e, McGraw-Hill Education (Italy) srl, ISBN 978-88-386-6587-5

[www.ateneonline.it/atzeni/](http://www.ateneonline.it/atzeni/)

I diritti di traduzione, di riproduzione, di memorizzazione elettronica e di adattamento totale e parziale con qualsiasi mezzo (compresi i microfilm e le copie fotostatiche) sono riservati per tutti i paesi.

Le riproduzioni effettuate per finalità di carattere professionale, economico o commerciale o comunque per uso diverso da quello personale possono essere effettuate a seguito di specifica autorizzazione rilasciata da CLEARedi, Corso di Porta Romana 108, Milano 20122, e-mail: [info@clearedi.org](mailto:info@clearedi.org) e sito web: [www.clearedi.org](http://www.clearedi.org)

Nomi e marchi citati sono generalmente depositati o registrati dalle rispettive case produttrici.

Publisher: Paolo Roncoroni  
Executive Editor: Filippo Aroffo  
Development Editor: Barbara Ferrario  
Realizzazione editoriale per il digitale: Giuseppe Bonelli  
Grafica di copertina: Feel Italia, Milano  
Immagine di copertina: ©Kentoh

ISBN: 9788838691126

# Sommario

<b>Appendice A</b> <b>Microsoft Access</b>	<b>3</b>
<b>Appendice B</b> <b>Il sistema DB2</b>	<b>23</b>
<b>Appendice C</b> <b>DBMS open source: Postgres</b>	<b>43</b>



# Prefazione

Le tre appendici che vengono qui offerte erano originariamente parte dei volumi di "Basi di dati" che sono stati pubblicati dagli autori a partire dal 1996. Le appendici descrivono le caratteristiche di alcuni sistemi per la gestione di basi di dati di grande rilievo: Microsoft Access, IBM DB2 e Postgres. Le appendici sono state aggiornate in corrispondenza del rilascio della quarta edizione del volume.

L'appendice dedicata a Microsoft Access descrive le caratteristiche principali di quello che è probabilmente il DBMS usato come piattaforma per il maggior numero di applicazioni gestionali, particolarmente nell'ambito di quelle di piccole dimensioni. Microsoft Access è uno dei componenti della famiglia di prodotti Microsoft Office e ciò garantisce un'ampia diffusione dello strumento. L'appendice è stata aggiornata alla versione Microsoft Access 2013 e descrive come si definisce una semplice basi di dati e si costruiscono interrogazioni in Access. In particolare, si mostra l'uso del linguaggio Query-By-Example per la formulazione di query in modo interattivo, senza la necessità di conoscere la sintassi di un linguaggio testuale.

L'appendice dedicata a IBM DB2 mostra alcune delle caratteristiche di uno dei maggiori DBMS commerciali, usato in modo molto esteso negli ambiti di mercato che ancora oggi fanno riferimento in modo significativo alle soluzioni hardware e software di IBM. L'appendice è aggiornata alla versione 8.2 e illustra il ricco insieme di strumenti di amministrazione di IBM DB2, cruciali per la gestione di un DBMS nell'ambito di un sistema informativo di grandi dimensioni. L'appendice descrive poi le modalità di gestione del dialogo con le applicazioni, fornendo una descrizione concreta dei principi illustrati nel libro di testo. Infine, vengono mostrate alcune delle varianti del linguaggio SQL che caratterizza IBM DB2 e si passano in rassegna alcune delle funzionalità avanzate del prodotto.

L'appendice dedicata a Postgres ha il ruolo di descrivere uno dei sistemi open-source più interessanti, dal punto di vista della varietà di tecniche adottate nella sua costruzione e dal punto di vista del supporto al linguaggio SQL offerto dal sistema. Il ruolo dei DBMS open-source è oggi molto rilevante ed è destinato a crescere di importanza in futuro. L'appendice illustra brevemente il modo in cui si può installare Postgres. Si descrivono poi gli strumenti per l'amministrazione del sistema, considerando sia moduli che prevedono un'interazione di tipo testuale, sia moduli che offrono un'interazione grafica. Si mostra quindi la costruzione di una base di dati d'esempio.

L'appendice quindi analizza le caratteristiche del dialetto SQL riconosciuto da Postgres e si conclude con la breve descrizione delle caratteristiche evolute che il sistema riesce a gestire, quali la presenza di un'estensione procedurale, l'integrazione tra SQL e altri linguaggi di programmazione e il supporto per i trigger.

*Gli Autori*



# Appendice A

## Microsoft Access

Access, prodotto dalla Microsoft, è il più diffuso sistema di gestione di basi di dati per l'ambiente Microsoft Windows. Access può essere utilizzato in due modalità: come gestore di basi di dati autonomo su personal computer e come interfaccia verso altri sistemi.

Come gestore di basi di dati autonomo risente dei limiti dell'architettura dei personal computer: offre un supporto transazionale limitato, con meccanismi di sicurezza, protezione dei dati e gestione della concorrenza piuttosto semplici e incompleti. D'altra parte ha un costo assai ridotto, che giustifica queste limitazioni, e le applicazioni cui è destinato non hanno tipicamente bisogno di fare un uso sofisticato di questi servizi. L'interfaccia del sistema sfrutta le potenzialità dell'ambiente grafico e offre un ambiente facile da usare, sia per l'utente applicativo sia per il progettista della base di dati. Quando è usato come client di database server relazionali, Access mette a disposizione le proprie funzionalità di dialogo per l'interazione con questi sistemi. Access, in questo contesto, può essere visto come uno strumento che permette di evitare di scrivere codice SQL, in quanto acquisisce schemi e semplici interrogazioni tramite una rappresentazione grafica facilmente comprensibile; questi input vengono tradotti in opportuni comandi SQL in modo trasparente. Il protocollo ODBC, descritto nel Capitolo 9, viene usato per la comunicazione tra Access e il database server.

La descrizione di Access si concentrerà sulle funzioni di gestore di basi di dati, ponendo particolarmente l'accento sulle funzionalità di definizione di schemi e interrogazioni. La presentazione si baserà sulla versione italiana di Access 2013, normalmente disponibile come un componente di Microsoft Office. Per una descrizione completa del sistema, invitiamo a consultare i manuali forniti con il programma e la guida in linea raggiungibile dal menu di aiuto (in alto a destra, caratterizzato dall'etichetta '?'). Sono anche facilmente reperibili un gran numero di testi dedicati alla descrizione di questo sistema.

### A.1 Caratteristiche del sistema

Il sistema viene attivato nel modo tradizionale in cui partono le applicazioni Windows, ovvero selezionando l'icona del programma in una finestra o in un menu.

All'avvio, il programma chiede se si vuole creare un nuovo database o aprire un database esistente, fornendo una lista dei database che sono stati utilizzati precedentemente dal programma. La creazione di un database può partire da un database vuoto o seguire una serie di modelli predefiniti (utili per utenti inesperti). A ogni database corrisponde un file; per aprire un database preesistente bisogna selezionare il corri-

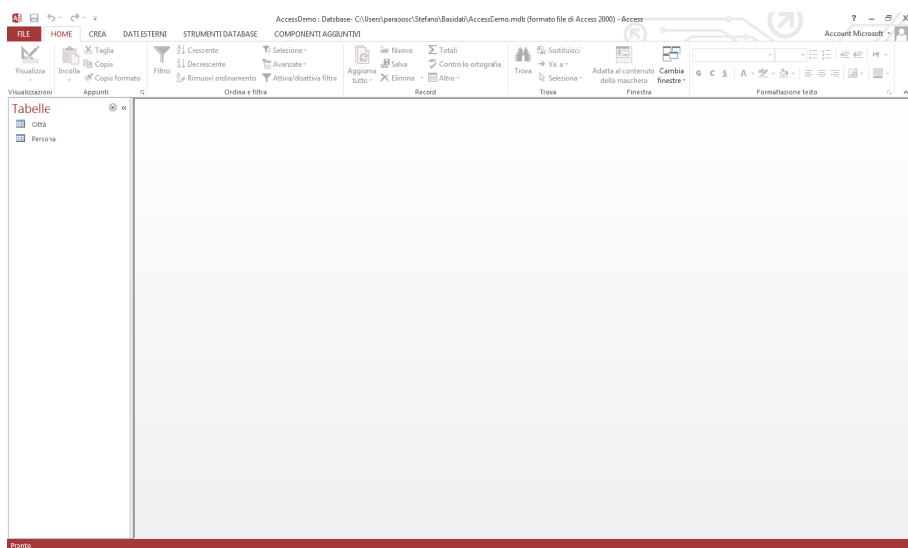


spontaneo file. Per creare un nuovo database o aprirne uno preesistente si possono anche utilizzare i comandi Nuovo e Apri del menu FILE dell'applicazione. La finestra principale è coerente con la struttura delle applicazioni Office 2013, presentando una serie di ambiti di lavoro, ciascuno caratterizzato da un opportuno insieme di comandi accessibili tramite icone poste sotto la classica barra dei menu. Nel caso di Access 2013, gli ambiti sono "Home", "Crea", "Dati esterni", "Strumenti database", "Campi" e "Tabella". A seconda del profilo di installazione del sistema operativo Windows, può comparire anche un'opzione "Componenti aggiuntivi". La Figura A.1 mostra l'aspetto dell'applicazione quando è selezionato l'ambito "Home" sulla base di dati di esempio. La finestra base dell'applicazione presenta nella configurazione standard sul lato sinistro tutti i nomi delle tabelle presenti nel database (se vi sono più tabelle di quante la finestra ne possa contenere, viene automaticamente aggiunta una barra di scorrimento con la quale è possibile navigare lungo la lista). In Figura A.1 si vede l'insieme di elementi della famiglia *Tabelle* presenti nella base di dati di riferimento. Osserviamo che la base di dati contiene una tabella CITTÀ e una tabella PERSONA.

Descriveremo ora la definizione di tabelle e interrogazioni, mentre tratteremo sommariamente le funzionalità offerte per la gestione di maschere, report, macro e moduli.

## A.2 La definizione delle tabelle

Per definire lo schema di una nuova tabella bisogna selezionare il contesto "Crea" dalla toolbar, che presenta tra le prime opzioni nel menu a icone sottostante proprio quelle relative alla creazione di nuove tabelle. Access propone la scelta tra diverse modalità di definizione della tabella, tra cui Tabella, Modelli di tabella e Struttura tabella. Con

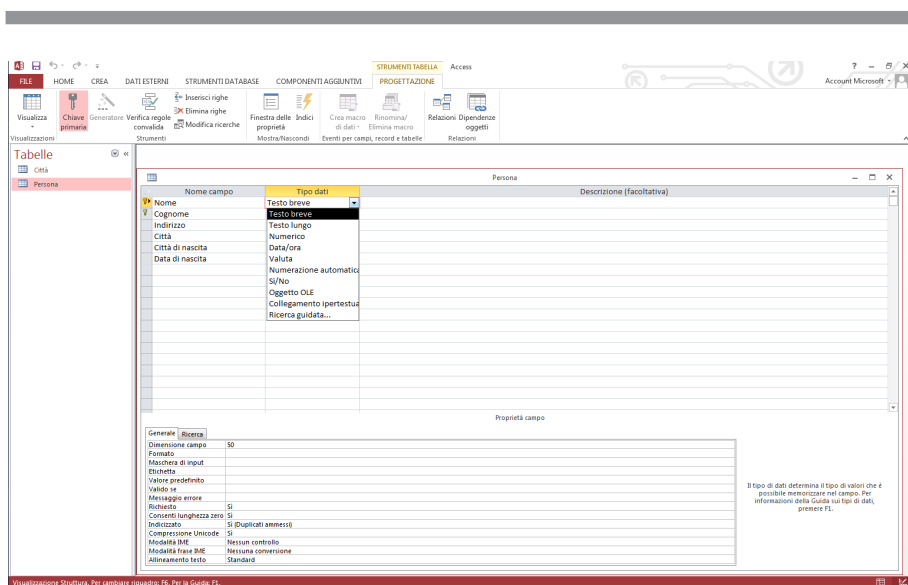


**Figura A.1** La finestra principale dell'applicazione

L'opzione **Tabella** si definisce lo schema della tabella come se si definisse la struttura di un *foglio elettronico (spreadsheet)*, ossia gli oggetti gestiti da applicazioni come Microsoft Excel. Questa interfaccia è dedicata a chi ha esperienza nell'uso di fogli elettronici e può rendere più facile per questa categoria d'utenti l'uso di un sistema relazionale. L'opzione **Modelli di tabella** consente l'uso di un "wizard", ovvero uno strumento di supporto che guida nella creazione di tabelle fornendo una collezione di esempi cui ispirarsi. Access propone l'uso dei wizard anche negli altri contesti di definizione dei vari altri componenti della base di dati. L'uso di wizard è consigliabile per la creazione di componenti dalla struttura regolare oppure nel caso di utenti poco esperti; per la gestione di situazioni particolari o nel caso di utenti sofisticati conviene invece fare uso delle funzionalità complete di definizione disponibili con la opzione **Struttura tabella**, su cui concentriamo l'analisi.

Per prima cosa si definiscono gli attributi, utilizzando la finestra che compare in Figura A.2. Per ciascun attributo bisogna specificare il nome e il dominio. Vi è poi un insieme di ulteriori informazioni definibili, che variano a seconda del dominio associato all'attributo (nel programma gli attributi vengono chiamati *campi*, i domini *tipi di dati*). I domini che possono essere associati a un attributo ricordano i domini dello standard SQL, con qualche arricchimento. I domini definibili sono i seguenti.

- **Testo breve:** permette di rappresentare stringhe di caratteri (corrisponde al dominio `varchar` di SQL).
- **Testo lungo:** permette di rappresentare stringhe di testo lunghe fino a 64.000 caratteri; non è un dominio esplicitamente previsto da SQL, ma può essere inteso come un caso particolare di `varchar`.
- **Numerico:** rappresenta la famiglia di domini numerici, interi e reali. Le proprietà del dominio permettono di specificare se l'attributo rappresenta valori precisi o ap-



**Figura A.2** Finestra di definizione dello schema delle tabelle

prossimati, e il grado di precisione della rappresentazione. Corrisponde alle famiglie di domini `numeric`, `decimal`, `integer`, `smallint`, `float`, `double` e `real`.

- **Data/Ora:** rappresenta istanti temporali e corrisponde ai domini SQL `date`, `time` e `timestamp`, a seconda che si decida di rappresentare rispettivamente solo la data, solo l'ora o entrambe.
- **Valuta:** serve per rappresentare valori monetari. È essenzialmente un caso particolare del dominio `Numerico`, caratterizzato da una rappresentazione numerica esatta su 8 byte con due cifre decimali. È rappresentato con un dominio apposito a causa della particolare rilevanza applicativa che ha la rappresentazione di valori monetari.
- **Numerazione automatica:** è un particolare dominio che associa a ogni riga della tabella un valore unico. Questo dominio permette di associare una chiave compatta alla tabella.
- **Sì/No:** corrisponde al dominio `boolean` di SQL-3.
- **Oggetto OLE:** rappresenta un generico oggetto che può essere gestito tramite OLE (*Object Linking and Embedding*). OLE è un protocollo che permette di specificare quale applicazione deve gestire un oggetto nell'ambiente Windows. In questo modo si possono inserire all'interno di una base di dati informazioni dei tipi più vari, come documenti di un word processor, fogli elettronici, immagini o informazioni multimediali, lasciando all'applicazione invocata tramite OLE il compito di gestirne adeguatamente il contenuto.
- **Collegamento ipertestuale:** permette la definizione di un riferimento. Il riferimento può essere interno alla base di dati o anche esterno (per esempio l'URL di una risorsa disponibile in Internet).
- **Ricerca guidata...**: questa opzione permette la definizione di un meccanismo che in fase di inserimento consente di scegliere un valore tra quelli presenti in un elenco predefinito o estratti tramite una query dal database.

Per ciascun attributo si possono poi specificare un certo numero di proprietà, che qualificano ulteriormente il suo dominio e la sua rappresentazione interna. Le proprietà compaiono nella parte inferiore della finestra di definizione dello schema.

- **Dimensione campo:** rappresenta la dimensione dell'attributo. Si può specificare solo per i domini `Testo breve` e `Numerico`. Per il tipo `Testo breve` la dimensione è un valore che rappresenta la lunghezza massima della stringa di caratteri. Per il tipo `Numerico`, si possono specificare le seguenti dimensioni:
  - **Byte:** interi su 8 bit (valori tra 0 e 255);
  - **Intero:** interi su 16 bit (valori tra -32768 e 32767);
  - **Intero lungo:** interi su 32 bit;
  - **Precisione singola:** rappresentazione in virgola mobile su 32 bit;
  - **Precisione doppia:** rappresentazione in virgola mobile su 64 bit;
  - **ID replica:** identificatore su 128 bit, unico per ogni tupla, anche in un sistema distribuito;
  - **Decimale:** rappresentazione in virgola mobile, con la possibilità di specificare il numero di cifre decimali significative.
- **Formato:** descrive il formato di visualizzazione dei valori degli attributi. Access utilizza dove possibile i valori specificati nell'ambiente Windows (opzioni di internazionalizzazione del pannello di controllo). Per la rappresentazione di date,

numeri e valori booleani, permette di scegliere tra vari formati predefiniti (sette formati predefiniti per le date, sette per i numeri, tre per i booleani). Si possono poi definire altri formati. È possibile inoltre distinguere la rappresentazione dei valori a seconda che siano positivi, negativi, pari a zero, o abbiano associato il valore nullo.

- **Precisione:** questa informazione, definibile solo per attributi di dominio numerico `Decimale` specifica quante cifre decimali devono essere utilizzate nella rappresentazione.
- **Scala:** questa informazione, definibile solo per attributi di dominio numerico `Decimale` specifica quante cifre decimali devono essere utilizzate al massimo a destra del separatore decimale.
- **Posizioni decimali:** questa informazione, definibile solo per attributi di dominio numerico `Precisione singola`, `Precisione doppia` e `Decimale`, specifica il numero di cifre da utilizzare a destra del separatore decimale.
- **Maschera di input:** questo parametro specifica il formato che deve essere utilizzato per l'immissione dei dati. Se per esempio un attributo registra un numero di telefono, composto da un prefisso di 3 cifre e da un numero di 7 cifre separati da un trattino, è possibile specificare una maschera di ingresso che individui le due parti e introduca immediatamente il trattino, permettendo all'utente di inserire solamente le cifre. Access mette a disposizione un wizard anche per la creazione della *Maschera di input*.
- **Etichetta:** rappresenta il nome che può essere dato all'attributo quando compare in una maschera o in un report. Può capitare di utilizzare come nome dell'attributo un nome compatto con cui sia possibile scrivere interrogazioni concise, mentre per la visualizzazione dei risultati si preferisca usare un nome che rappresenti meglio il significato dell'attributo.
- **Valore predefinito:** con questo parametro si specifica il valore di default per l'attributo. Corrisponde esattamente all'opzione `default` di SQL. Tutte le volte che si inserisce una nuova tupla, il valore di default comparirà come valore per l'attributo. Si può utilizzare come valore di default anche il risultato di un'espressione, come per esempio `=Date()` che assegna a un campo di dominio `Data/Ora` il giorno corrente.
- **Valido se:** permette di specificare un vincolo sull'attributo. Access verifica automaticamente che ogni valore inserito appartenga al dominio dell'attributo. Oltre a questo controllo, Access permette di specificare un vincolo generico per ogni attributo (in modo analogo alla clausola `check` di SQL). Questo vincolo viene espresso utilizzando la sintassi che è utilizzata per la specifica delle condizioni in QBE, che vedremo nei prossimi paragrafi.
- **Messaggio errore:** specifica il messaggio che deve essere visualizzato quando viene inserito un valore che non rispetta il vincolo d'integrità.
- **Richiesto:** specifica se deve essere sempre presente un valore per l'attributo. La proprietà può essere vera o falsa e corrisponde al vincolo `not null` di SQL.
- **Consenti lunghezza zero:** è una proprietà che vale solo per gli attributi di tipo `Testo breve` e `Testo lungo`. Specifica se devono essere ammesse stringhe di lunghezza nulla, o se una stringa di lunghezza nulla deve essere considerata come un valore nullo. A seconda dei contesti applicativi, può essere utile gestire in modo

diverso le stringhe vuote dal valore nullo. Si tenga presente che il valore nullo viene trattato in modo particolare da SQL, per cui un confronto di disuguaglianza su stringhe è soddisfatto da una stringa di lunghezza zero, ma non da una stringa nulla.

- **Indicizzato:** specifica se deve essere costruito un indice sull'attributo. Le opzioni possibili sono No, Sì (Duplicati ammessi) e Sì (Duplicati non ammessi). La terza opzione definisce sull'attributo un indice di tipo *unique*. Questo è anche il modo in cui si rappresentano i vincoli di tipo *unique*. Non si possono definire indici su attributi Oggetto OLE. Con questa modalità si permette solo la definizione di indici su un solo attributo; per la definizione di indici più complicati bisogna operare al livello di tabella.
- **Compressione Unicode:** specifica per i campi di tipo testuale se la rappresentazione Unicode a 16 bit dei caratteri, che costituisce lo standard di Microsoft Access a partire dalla versione 2000, deve essere compressa nei casi in cui si rappresentano caratteri del normale alfabeto latino. La compressione avviene omettendo la rappresentazione del primo byte, il quale è pari a zero per tutti i caratteri del dominio ASCII.
- **Modalità IME:** specifica se sul campo deve essere utilizzato un *Input Method Extension*, ovvero una modalità particolare di acquisizione dell'input che consente di gestire gli alfabeti ideogrammatici che caratterizzano le lingue orientali. Non è un'opzione significativa se non si prevedono di utilizzare alfabeti di questo tipo.
- **Modalità frase IME:** specifica ulteriormente la modalità di input da adottare per l'acquisizione di frasi nei linguaggi ideogrammatici.
- **Smart tag:** specifica se all'attributo deve essere associata un'etichetta intelligente, che permette di invocare funzioni predefinite nella fase di input e consultazione del campo (per esempio, per attributi che rappresentano date è possibile accedere a un calendario o invocare l'applicazione di gestione dell'agenda).

Dopo aver definito i vari attributi, si completa la sessione di definizione indicando quali attributi devono essere considerati la chiave primaria della tabella. L'indicazione degli attributi di chiave avviene selezionando gli attributi e premendo il pulsante che raffigura la chiave nella barra degli strumenti associata all'opzione "Progettazione". Gli attributi che costituiscono la chiave vengono visualizzati con un'icona che rappresenta una chiave nella colonna che precede il nome. Automaticamente Access definirà un indice di tipo *unique* sugli attributi che costituiscono la chiave.

Si possono poi definire ulteriori proprietà a livello di tabella (cui si accede tramite l'icona Finestra delle proprietà associata all'opzione "Progettazione". Le più significative sono le seguenti.

- **Descrizione:** è una descrizione testuale del contenuto della tabella.
- **Valido se:** specifica un vincolo che deve essere soddisfatto da ogni tupla della tabella. A livello di tabella si definiscono vincoli che coinvolgono diversi attributi. Pure in questo caso, la sintassi è quella usata per l'espressione delle condizioni sulle interrogazioni. Il controllo viene effettuato al termine dell'inserimento di ogni tupla.
- **Messaggio errore:** rappresenta il messaggio che viene visualizzato quando il sistema rileva una violazione del vincolo.
- **Filtro:** specifica le condizioni che devono essere soddisfatte dagli elementi che si vogliono visualizzare.

- **Ordina per:** descrive gli attributi rispetto ai quali si devono ordinare le tuple della tabella.

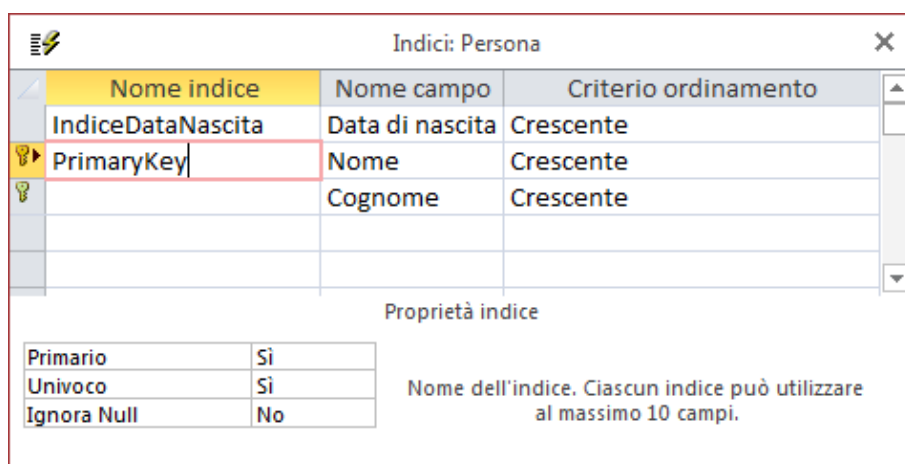
Per specificare indici su più attributi si deve aprire la finestra di definizione degli indici, premendo il pulsante INDICI sulla barra degli strumenti nell'opzione "Progettazione". La finestra contiene una tabella (mostrata in Figura A.3) con colonne Nome indice, Nome campo e Criterio ordinamento. Per definire un indice su più attributi, si inserisce in una riga il nome dell'indice, il nome del primo attributo e la direzione di ordinamento. Nella riga successiva si lascia vuoto il nome dell'indice e si introduce il nome del secondo attributo e la corrispondente direzione di ordinamento, proseguendo allo stesso modo per tutti gli attributi che caratterizzano l'indice.

Prima di terminare la sessione di definizione bisogna salvare il risultato, chiudendo per esempio la finestra di definizione della struttura. Access a questo punto chiede quale nome deve essere associato alla tabella. Il nome può anche contenere al suo interno degli spazi.

## A.2.1 Specifica dei cammini di join

Un cammino di join è un legame tra coppie di attributi di due tabelle e si usa per specificare che tra le due tabelle viene normalmente eseguito un join basato sull'uguaglianza di quegli attributi. Un cammino di join viene rappresentato graficamente da una linea che collega le due tabelle. Per esempio, nella base di dati "persone e città", esiste un cammino di join tra l'attributo Città di nascita di PERSONA e Nome di CITTA'. Access permette la definizione di cammini di join (chiamati *relazioni*); per ogni cammino di join è inoltre possibile specificare se vi è associato un vincolo di integrità referenziale. Tutto questo avviene senza che venga immesso del testo, operando esclusivamente in modo grafico. Descriviamo brevemente la procedura.

Si inizia selezionando l'opzione Relazioni dalla toolbar associata all'opzione "Strumenti database". Si apre a questo punto una finestra (Figura A.4) in cui si possono



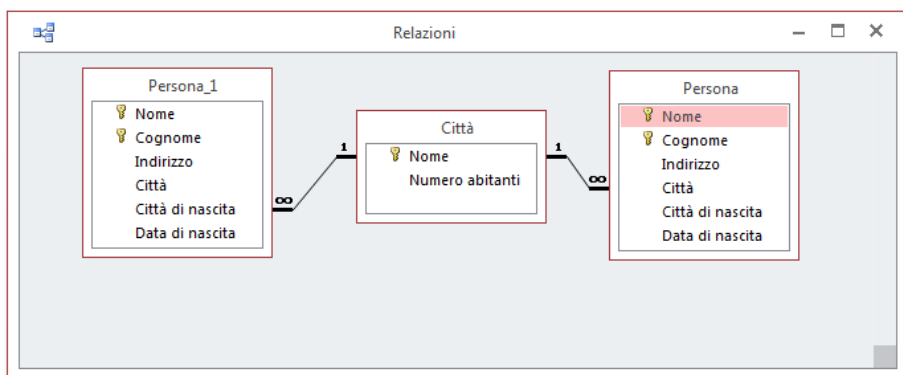
Indici: Persona		
Nome indice	Nome campo	Criterio ordinamento
IndiceDataNascita	Data di nascita	Crescente
PrimaryKey	Nome	Crescente
	Cognome	Crescente

Proprietà indice	
Primario	Sì
Univoco	Sì
Ignora Null	No

Nome dell'indice. Ciascun indice può utilizzare al massimo 10 campi.

**Figura A.3** La finestra di descrizione degli indici



**Figura A.4** La finestra di definizione dei cammini di join

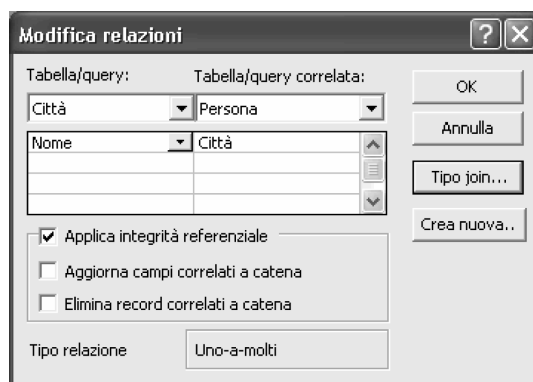
inserire gli schemi delle tabelle create, selezionandole da una lista. È possibile definire i cammini di join tra gli schemi selezionando un attributo di uno schema e, tenendo premuto il pulsante del mouse, spostando il puntatore del mouse sull'attributo corrispondente dell'altra tabella. Una volta definito il legame tra gli attributi, Access apre una finestra che rappresenta gli attributi coinvolti nel join e che permette di estendere la condizione di join ad altri attributi o di modificare la relazione così definita. Questa finestra è rappresentata in Figura A.5. Premendo il pulsante TIPO JOIN di questa finestra, si ha la possibilità di scegliere quale tipo di join, *inner*, *outer left* o *outer right*, deve essere utilizzato nel legame tra le due tabelle (a differenza di SQL, il *full outer join* non è offerto). In questo modo tutte le volte che si definisce una query che accede alle due tabelle, verranno assegnate di default le condizioni di join specificate.

Definito il cammino di join, è possibile specificare se a esso deve essere associato un vincolo di integrità referenziale. Nella stessa finestra, Access permette di definire una politica di reazione alle violazioni, per cui si può imporre che le modifiche o le cancellazioni siano seguite da corrispondenti modifiche e cancellazioni nelle altre tabelle. Se la politica non è specificata, ogni modifica che introduca una violazione viene semplicemente impedita.

Rispetto allo standard SQL-2, Access permette quindi un insieme di reazioni più limitato, corrispondente alle sole scelte di *cascade delete* e *cascade update*. Per specificare i vari vincoli di integrità referenziale, si devono seguire i criteri di progettazione descritti nel Capitolo 8. Access non permette che nel grafo compaia più di un cammino tra due tabelle; se si devono rappresentare più cammini di join tra due tabelle, bisogna introdurre più esemplari della stessa tabella del grafo (Figura A.4).

## A.2.2 Popolamento delle tabelle

Dopo aver definito lo schema delle tabelle, si possono inserire tuple nella tabelle per popolare l'istanza della base di dati. Anche per questo compito Access fornisce un'interfaccia grafica facile da usare. Aprendo una tabella dalla finestra di partenza selezionandola con doppio clic dall'elenco, compare una rappresentazione tabellare del contenuto della tabella. Questa consiste in una griglia con colonne con intestazione



**Figura A.5** La finestra di definizione delle proprietà dei join

pari ai nomi degli attributi, e righe che descrivono le tuple della tabella, cui si somma una riga vuota che serve per inserire nuove tuple nella tabella. Access aggiunge anche una colonna aggiuntiva, che consente di modificare lo schema aggiungendo nuovi attributi. La Figura A.6 rappresenta questa finestra.

L'inserimento avviene ponendo il cursore nell'ultima riga e digitando un valore per ogni attributo. Se il valore inserito non rispetta tutti i vincoli definiti sull'attributo, l'inserimento viene immediatamente rifiutato. Spostando il cursore al di fuori della riga, si indica implicitamente che l'inserimento è terminato. A questo punto il sistema controlla che tutti i vincoli siano rispettati, ovvero che i campi per cui è necessario fornire un valore siano stati specificati e che i valori degli attributi rispettino le regole di validazione definite. Per effettuare delle modifiche di attributi basta andare a selezionare il valore da modificare con il cursore e inserire il nuovo valore. Terminata l'immissione del valore e spostato il cursore in una diversa posizione, i vincoli sono verificati e la modifica è completata.

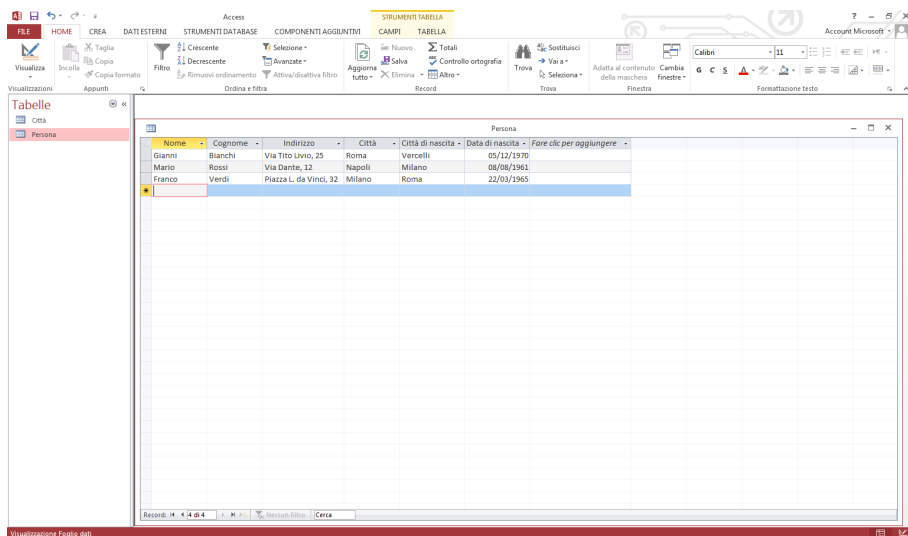
## A.3 La definizione di query

Per la definizione di interrogazioni, Access mette a disposizione due diversi strumenti: uno strumento grafico di formulazione di interrogazioni di tipo QBE (*Query By Example*) e un interprete SQL. Descriviamo dapprima le caratteristiche dell'interfaccia QBE, analizzando successivamente l'interprete SQL.

### A.3.1 Query By Example

Il nome QBE fa riferimento a una famiglia molto vasta di linguaggi di interrogazione per basi di dati, che realizzano un'implementazione delle idee di base del calcolo relazionale dei domini (Paragrafo 3.2.1). Il punto fondamentale è che un'interrogazione venga formulata descrivendo le caratteristiche che devono essere possedute dalle righe del risultato. La definizione di una query avviene riempiendo uno schema di tabella con tutti gli attributi e le condizioni che caratterizzano una riga "esemplare" del risultato.





**Figura A.6** La finestra che visualizza l'istanza della tabella

Per definire una nuova interrogazione si seleziona l'opzione "Crea" della finestra di base dell'applicazione e si preme il pulsante *Struttura query*, che apre la finestra di progettazione di query. Questa finestra è divisa in due metà. La metà superiore è inizialmente vuota e viene riempita con una descrizione degli schemi delle tabelle selezionate da una lista. Le tabelle sono collegate dai cammini di join predefiniti (possiamo in effetti interpretare la metà superiore della finestra come la porzione del diagramma Relazioni rilevante per l'interrogazione che si deve definire). Nella metà inferiore della finestra compare una tabella inizialmente vuota, con un insieme di colonne senza nome e con righe etichettate Campo, Ordinamento, Mostra e Criteri.

Le celle della riga Ordinamento possono essere vuote o contenere una tra le opzioni Crescente o Decrescente. Quando la riga non è vuota viene imposto un ordinamento delle tuple del risultato, secondo i valori dell'attributo associato alla colonna in cui compare l'opzione. Se vi sono più colonne con la cella Ordinamento attivata, si applica per primo l'ordinamento che compare più a sinistra nelle colonne; a pari valori dell'attributo, si applicheranno man mano le successive condizioni di ordinamento, procedendo da sinistra a destra.

Le celle della riga Mostra contengono un quadratino che può o meno contenere un segno di spunta. Se il quadratino contiene tale segno, l'attributo che compare nella colonna dovrà far parte del risultato della interrogazione. Lo stato del quadratino commuta con un clic del mouse.

Le celle della riga Criteri contengono le condizioni che devono essere soddisfatte dalle tuple risultato della interrogazione. Le condizioni possono essere semplici confronti tra il valore dell'attributo che compare in cima alla colonna e una costante; in questo caso basta inserire il valore della costante nella cella. La condizione può anche essere più complicata e includere confronti più ricchi, espressioni e riferimenti ad altri attributi, come vedremo negli esempi successivi.

Per mettere i nomi degli attributi in cima alle colonne si possono utilizzare due modalità: si può scrivere direttamente il nome dell'attributo, eventualmente qualificato col nome della tabella di appartenenza, o si possono selezionare gli attributi che compaiono nella rappresentazione degli schemi della metà superiore della finestra, "trascinandoli" nelle relative colonne o facendo un doppio clic su di essi (a quel punto l'attributo viene riportato nella prima colonna libera). Dopo che la query è stata formulata, per eseguirla bisogna premere il pulsante Esegui della barra degli strumenti, associato a un punto esclamativo; dopo l'esecuzione, la tabella risultato della query compare al posto della finestra di definizione della query.

Vediamo alcuni esempi di definizione di interrogazioni. Supponiamo di avere una base di dati con una tabella PERSONA (Nome, Cognome, Indirizzo, Città, Città di nascita, Data di nascita) e una tabella CITTÀ (Nome, Numero abitanti). Per trovare i nomi, ordinati alfabeticamente, delle persone aventi cognome Rossi possiamo riempire lo schema nel modo descritto in Figura A.7.

Quando sono riempiti più campi della riga Criteri, l'interrogazione considera tutte le condizioni in congiunzione. Se bisogna selezionare le tuple che soddisfano più condizioni in disgiunzione, bisogna riempire più righe con i diversi criteri. Access etichetta automaticamente con Oppure le righe aggiuntive. Così, per trovare i nomi, cognomi e indirizzi delle persone di Milano aventi cognome Rossi o Bianchi, si creerà uno schema come quello in Figura A.8.

Nella lista degli attributi di una tabella compare anche il simbolo di asterisco che, con significato analogo a SQL, rappresenta tutti gli attributi. Per selezionare quindi tutti gli attributi delle tuple della tabella PERSONA che hanno la residenza nella città di nascita, potremo formulare la query QBE rappresentata in Figura A.9.

In questa interrogazione, per imporre l'uguaglianza di due attributi, abbiamo imposto come valore dell'attributo Città il nome dell'attributo Città di nascita racchiuso tra pa-

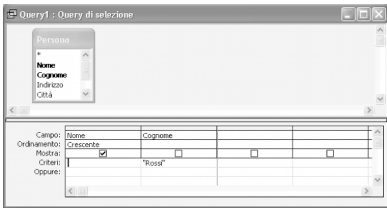
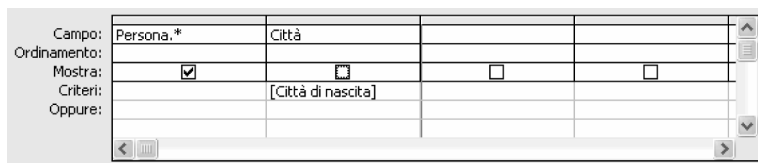


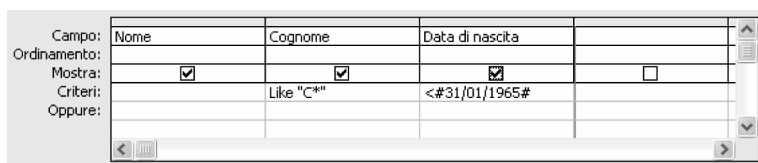
Figura A.7 Interrogazione QBE che restituisce i nomi delle persone di cognome Rossi

Campo:	Nome	Cognome	Indirizzo	Città
Ordinamento:	Crescente			
Mostra:	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Criteri:		"Rossi"		"Milano"
Oppure:		"Bianchi"		"Milano"

Figura A.8 Query che restituisce i milanesi chiamati "Rossi" o "Bianchi"



**Figura A.9** Query che restituisce le persone con città di nascita e di residenza uguali

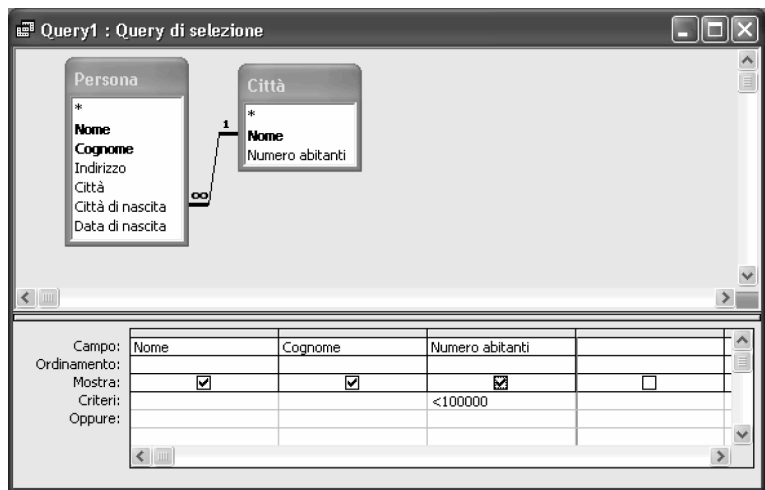


**Figura A.10** Query che restituisce i nomi, i cognomi e le date di nascita delle persone con un cognome che inizia per 'C' nate prima del 31/1/65

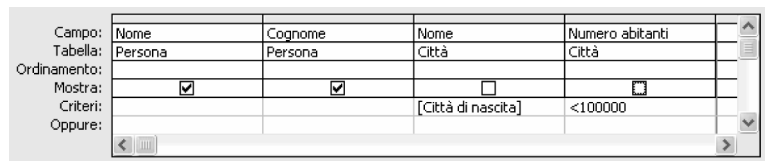
rentesi quadre. Le parentesi quadre sono il costrutto sintattico che permette ad Access di distinguere le stringhe di caratteri costanti dai riferimenti ai componenti dello schema. Access consente poi di formulare delle condizioni utilizzando i normali operatori di confronto (<, ≤, >, ≥ e <>) e l'operatore *Like*, per il confronto di stringhe con espressioni regolari che usano i caratteri speciali \* e ? (che corrispondono rispettivamente ai caratteri % e \_ della sintassi standard SQL). Per trovare i nomi, i cognomi e le date di nascita delle persone che sono nate prima del 31 gennaio 1965 e che hanno un cognome che inizia con la lettera 'C', si potrà formulare la query mostrata in Figura A.10.

A ogni interrogazione è possibile associare delle proprietà, a diversi livelli. A livello di singola colonna, si può specificare un formato di visualizzazione diverso da quello immesso nella fase di definizione dello schema della tabella. Un'altra importante proprietà è l'eliminazione di eventuali duplicati presenti nel risultato. Per specificare che i duplicati devono essere rimossi bisogna accedere alla finestra di descrizione delle proprietà della query e assegnare alla proprietà Valori univoci il valore Sì.

Per formulare delle interrogazioni che richiedono più tabelle, è opportuno far comparire nella finestra superiore le tabelle richieste (selezionandole dalla finestra di dialogo che compare appena si crea una nuova interrogazione). Le tabelle compariranno legate dai cammini di join che sono stati definiti al momento della definizione dello schema. Nella tabella nella metà inferiore della finestra di definizione della query è possibile aggiungere anche la riga Tabella (Figura A.12), selezionando l'opzione Nomi tabelle dalle icone nella toolbar. La riga rappresenta il nome della tabella da cui viene prelevato l'attributo. Le condizioni di join tra le tuple delle tabelle non dovranno essere specificate se queste sono state predefinite. Quando le condizioni di join predefinite non sono quelle richieste dalla particolare query, è possibile modificare i cammini che collegano le tabelle intervenendo sulla rappresentazione grafica della parte superiore della finestra. Volendo per esempio formulare la query che trova le persone nate in



**Figura A.11** Query che restituisce i nomi, i cognomi e le date di nascita delle persone nate in una città con meno di 100 000 abitanti (con join predefinito)



**Figura A.12** Query che restituisce i nomi e cognomi delle persone nate in una città con meno di 100.000 abitanti (senza join predefinito)

una città con meno di centomila abitanti, se è stato predefinito il legame di join tra le tabelle PERSONA e CITTÀ, si può formulare la query che appare in Figura A.11. Qualora invece il legame di join non sia stato predefinito, si dovrà rendere esplicita la condizione nella tabella, formulando un'interrogazione come quella in Figura A.12, che usa i riferimenti agli attributi nelle righe Criteri.

Può essere necessario talvolta introdurre tra le tabelle che compaiono nella metà superiore anche tabelle di cui non vengono utilizzati attributi nella query. Un caso importante di questo tipo è quello in cui si devono estrarre informazioni tra due tabelle che non posseggono un cammino di join diretto, ma in cui il join è realizzato tramite una tabella intermedia. In questo caso, anche se l'interrogazione utilizzerà per la visualizzazione e l'applicazione di condizioni solo attributi delle due tabelle esterne, la tabella intermedia dovrà comparire nella metà superiore, con un approccio simile a quello utilizzato dall'algebra, dal calcolo relazionale (Capitolo 3) e da SQL (Capitolo 4). Infatti, senza un cammino di join che leghi le tabelle, il sistema esegue il prodotto cartesiano delle tabelle che compaiono nella metà superiore, applicando poi le condizioni che sono state definite per la query.

Vediamo ora la formulazione di interrogazioni QBE facenti uso di operatori aggregati. Gli operatori aggregati che Access mette a disposizione sono Somma, Media, Min, Max, Conteggio, che corrispondono agli operatori standard SQL `sum`, `avg`, `min`, `max` e `count`. A questi si aggiungono gli operatori `DevSt` (la deviazione standard), `Var` (la varianza), `Primo` e `Ultimo` (il valore dell'attributo rispettivamente per la prima e per l'ultima tupla). Per utilizzare questi operatori è necessario introdurre una nuova riga nello schema della query, la riga Formula. Questa riga viene introdotta selezionando l'opzione Totali dalla toolbar. Vediamo un semplice esempio d'uso degli operatori, definendo, come mostrato in Figura A.13, una query che permette di trovare il numero di tuple presenti nella tabella PERSONA.

Il fatto che nella riga Formula compaia il valore Conteggio fa sì che il risultato della interrogazione non sia l'elenco dei valori dell'attributo Nome della tabella PERSONA, ma appunto il numero di tuple. Nella riga Campo deve comparire uno qualsiasi degli attributi della tabella PERSONA; sarebbe stato meglio, per coerenza con SQL-2, che Access permettesse in questo contesto l'uso dell'asterisco (\*), utilizzando invece il nome di un attributo per contare i diversi valori dell'attributo; purtroppo nella valutazione degli altri operatori aggregati l'asterisco crea delle complicazioni, per cui in Access si è scelta la soluzione di impedire l'uso dell'asterisco quando è abilitata la riga Formula.

Nella interrogazione in Figura A.14 l'attributo Cognome viene caratterizzato dal valore Raggruppamento nella riga Formula, ovvero l'attributo viene utilizzato per raggruppare le tuple della tabella PERSONA. La seconda colonna rappresenta l'applicazione dell'operatore Conteggio a ogni singolo raggruppamento. Access permette di esprimere delle condizioni sul risultato della valutazione degli operatori aggregati, in modo analogo alla clausola `having` in SQL. Per fare ciò, è sufficiente scrivere le condizioni nella riga Criteri, in modo analogo all'espressione di semplici condizioni sulle tuple. Così, per trovare i cognomi che sono posseduti da almeno due persone, si potrà scrivere l'interrogazione QBE in Figura A.15, che restituisce i cognomi posseduti da più di una persona, indicando per ogni cognome il numero di volte che questo appare nella tabella PERSONA.

Se in una interrogazione con raggruppamento le tuple devono essere selezionate preliminarmente in base ai valori di attributi che non vengono utilizzati nel raggruppamento, diventa necessario distinguere le condizioni che devono essere valutate prima del raggruppamento da quelle che devono essere eseguite nella fase successiva. Questa distinzione in SQL avviene ponendo le condizioni preliminari nella clausola `where` e le condizioni successive nella clausola `having`; nel linguaggio QBE la distinzione

Campo:	Nome	
Tabella:	Persona	
Formula:	Conteggio	
Ordinamento:		
Mostra:	<input checked="" type="checkbox"/>	
Criteri:		
Oppure:		

**Figura A.13** Query che restituisce il numero di persone

Campo:	Cognome	Nome		
Tabella:	Persona	Persona		
Formula:	Raggruppamento	Conteggio		
Ordinamento:				
Mostra:	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Criteri:				
Oppure:				

**Figura A.14** Query che restituisce il numero di persone che possiedono ciascun cognome

Campo:	Cognome	Nome		
Tabella:	Persona	Persona		
Formula:	Raggruppamento	Conteggio		
Ordinamento:				
Mostra:	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Criteri:		>1		
Oppure:				

**Figura A.15** Query che restituisce il numero di persone che possiedono ciascun cognome, per i cognomi posseduti da più persone

avviene ponendo il valore `Dove` nella riga Formula per gli attributi che servono solo per la selezione delle tuple da raggruppare.

La presenza del valore `Dove` è incompatibile con il flag di `Mostra`. Infatti, come in SQL, nel risultato delle interrogazioni che fanno uso di operatori aggregati possono comparire solo il risultato della valutazione degli operatori aggregati e gli attributi su cui viene effettuato il raggruppamento. Per esempio, per trovare le persone nate dopo l'1/1/75 che hanno lo stesso cognome di persone nate dopo la stessa data, si può utilizzare l'operatore `Count` e formulare l'interrogazione in Figura A.16.

Ci sono dei modi alternativi per formulare questa interrogazione che non fanno uso del termine `Dove`. Uno di questi consiste nel formulare e salvare nella base di dati un'interrogazione preliminare che estrae solo le persone nate dopo l'1/1/75; si può poi definire una seconda interrogazione, che parte dal risultato della prima andando a fare un raggruppamento in base al valore di `Cognome`, identificando così gli omonimi. A ogni interrogazione, infatti, Access associa un nome e un'interrogazione può estrarre le informazioni sia dalle tabelle del database, sia dalle interrogazioni già definite. Ogni interrogazione che viene salvata e resa persistente può quindi essere considerata come una definizione di una vista sul database.

Campo:	Cognome	Data di nascita	Cognome	
Tabella:	Persona	Persona	Persona	
Formula:	Raggruppamento	Dove	Conteggio	
Ordinamento:				
Mostra:	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Criteri:		>#01/01/1975#	>1	
Oppure:				

**Figura A.16** Query che restituisce gli omonimi tra le persone nate dopo l'1/1/75

### A.3.2 L'interprete SQL

Oltre al linguaggio di interrogazione QBE, Access fornisce un interprete SQL, che può essere usato in alternativa a QBE. Access permette di passare rapidamente dal contesto QBE al contesto SQL e viceversa, scegliendo l'opzione **VISUALIZZAZIONE SQL** dalla toolbar. Il passaggio da un ambiente all'altro trasforma l'interrogazione corrente nella corrispondente interrogazione nell'altro ambiente.

Il passaggio da QBE a SQL è sempre possibile. In effetti, tutte le volte che una query QBE viene mandata in esecuzione, essa viene prima tradotta nella corrispondente forma SQL e quindi eseguita dall'interprete SQL. Il passaggio inverso non è invece sempre possibile, in quanto il linguaggio SQL è più potente di QBE, permettendo per esempio l'espressione di query con l'operatore di unione. Il linguaggio QBE è un linguaggio molto potente e facile da usare quando si devono formulare interrogazioni che fanno uso solo di selezioni, proiezioni e join: in questo caso la possibilità di formulare le interrogazioni senza bisogno di scrivere del testo secondo una rigida sintassi rappresenta un notevole aiuto. D'altra parte QBE non mette a disposizione un meccanismo adeguato per la rappresentazione di interrogazioni complesse, come quelle che richiedono di formulare delle interrogazioni nidificate in SQL. Di fatto, quando una interrogazione SQL che fa uso di query nidificate viene tradotta in QBE, la traduzione riporta semplicemente il testo dell'intera query nidificata nell'opportuna cella della riga Criteri.

Per quanto riguarda la sintassi riconosciuta dall'interprete, si tratta di un'estensione della sintassi standard SQL, con un supporto per particolari funzionalità e determinate differenze sintattiche e semantiche. Alcune delle differenze sono le seguenti:

- la clausola `top` può essere usata per selezionare un certo numero di tuple dal risultato;
- vengono usate le parentesi quadre per racchiudere gli identificativi di tabelle e attributi (necessarie quando compaiono degli spazi o caratteri speciali all'interno degli identificatori);
- l'operatore `join` deve essere sempre qualificato con il termine `inner` od `outer`;
- la valutazione dell'operatore `count` è diversa: se si dà come argomento un attributo, non vengono restituiti i distinti valori dell'attributo, bensì il numero di valori non nulli (come se fosse specificata implicitamente l'opzione `all`); l'opzione `distinct` non è riconosciuta.

Per esempio, si consideri la query QBE in Figura A.17, per cui è specificata la proprietà che vengano restituiti solo i primi 10 elementi e per cui è stato predefinito un cammino di join. A essa corrisponde la seguente query nel dialetto SQL di Access:

```
select top 10 Cognome, count(Cognome) as NumeroOmonimi
```

```
from Persona inner join Città
  on Persona.[Città di nascita] = Città.Nome
where [Numero abitanti] > 200000
group by Cognome
having count(Cognome) > 1
order by count(Cognome) desc
```

### A.4 Maschere e report

Le *maschere* permettono di rappresentare il contenuto del database in modo molto più chiaro e comprensibile, rispetto alla piatta rappresentazione delle righe delle tabelle.

Campo:	Cognome	Cognome	Numero abitanti	
Tabella:	Persona	Persona	Città	
Formula:	Raggruppamento	Conteggio	Dove	
Ordinamento:		Decrescente		
Mostra:	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Criteri:		>1	>200000	
Oppure:				

**Figura A.17** Query che restituisce il numero di omonimi in città sopra i 200.000 abitanti

Le maschere sono analoghe ai “moduli prestampati”, caratterizzati da un insieme di caselle in cui si devono inserire i dati, e un insieme di etichette che specificano quale dato deve essere inserito nella particolare casella. Le maschere possono servire per l’inserimento dei dati, realizzando una versione elettronica del modulo prestampato, e possono anche essere utilizzate per visualizzare e modificare il contenuto della base di dati.

Gli strumenti di generazione di maschere sono sempre stati uno degli strumenti di supporto più comuni tra quelli offerti dai DBMS commerciali. Access, al posto della semplice e tradizionale interfaccia a caratteri, sfrutta le prerogative dell’ambiente Microsoft Windows e permette la realizzazione di maschere grafiche. Lo strumento di definizione delle maschere permette di definire la posizione e il significato di ciascun componente della maschera, dando al progettista un’ampia libertà di personalizzazione a livello di fonti di caratteri, colori, disegni e simboli grafici.

Per creare una nuova maschera, bisogna selezionare il componente Maschere dalla toolbar associata all’opzione “Crea”. Il sistema offre sia uno strumento di progetto di base, sia uno strumento per la creazione guidata; mediante la creazione guidata si può creare in pochi istanti una semplice maschera associata a una tabella. Access offre in questo contesto la possibilità di scegliere tra vari strumenti per la creazione di maschere; essi differiscono in base allo schema di maschera che producono (per cui uno strumento produce una maschera con tutti gli attributi della tabella in sequenza uno per riga, un altro produce una rappresentazione tabellare arricchita da spazi e colori ecc.). Se non si utilizzano i servizi di uno strumento di creazione guidata, ci si trova davanti a una pagina bianca in cui si possono inserire man mano i componenti della maschera. La stessa interfaccia può essere usata per esplorare e modificare la struttura di una maschera preesistente, selezionando la maschera e premendo il pulsante STRUTTURA.

Una maschera è composta da diversi elementi, che devono essere definiti uno a uno. L’elemento di base di una maschera è il *controllo*, un oggetto cui corrisponde un’area rettangolare dello schermo e che può essere di tre tipi: *associato*, *non associato* e *calcolato*. Un controllo *associato* è un elemento della maschera cui viene associato un attributo di una tabella. Questo elemento rappresenterà il valore dell’attributo per la particolare tupla considerata. La rappresentazione del valore di un attributo in genere richiede di riprodurre una sequenza di caratteri; per attributi di tipo OLE, la rappresentazione viene affidata all’applicazione a esso associata. Un controllo *non associato* invece contiene un valore fisso, che non cambia al variare delle tuple; in questo modo si rappresentano tipicamente le etichette delle maschere (che anche in questo caso possono essere sequenze di caratteri, o altri oggetti di tipo generico; se si vuole



per esempio inserire un logo nella maschera, bisognerà associare a un controllo non associato la figura che rappresenta il logo). Infine, i controlli *calcolati* permettono di visualizzare i risultati di espressioni valutate su combinazioni arbitrarie di parametri costanti e valori degli attributi delle tuple. I controlli *calcolati* non possono essere usati per l'immissione o la modifica dei valori degli attributi.

Con una maschera è anche possibile inserire nuove tuple, riportando direttamente i valori degli attributi nei controlli *associati*. Con le maschere è anche possibile interrogare la base di dati, utilizzando un semplice comando di ricerca. Per modificare il valore di un attributo, si seleziona la tupla e si apporta la modifica direttamente sulla casella che presenta il valore dell'attributo. Le modifiche vengono rese persistenti spostandosi su una diversa tupla, tipicamente premendo i tasti per l'avanzamento o l'arretramento di una pagina che permettono di scandire le tuple della tabella. La Figura A.18 mostra una maschera sulle tuple di PERSONA, estese con un attributo che contiene una foto.

Le maschere sono in genere costruite su una singola tabella. È possibile definire delle maschere che ne contengono altre al loro interno, specificando dei legami tra i valori visualizzati nelle maschere. In questo modo si possono per esempio definire delle maschere per la gestione di tabelle con relazioni uno a molti, in cui la maschera interna rappresenta tutte le tuple della relazione di dettaglio associate all'elemento esterno (si pensi a una coppia di tabelle che descrivono gli ordini; una maschera può illustrare al livello più esterno i dati dell'ordine e in una sottomaschera può comparire la distinta degli ordinativi, appartenenti a due tabelle distinte).

Un *report* si definisce in modo analogo a una maschera, usando gli stessi strumenti e concetti. La differenza principale consiste nel fatto che un report ha normalmente l'obiettivo di fornire una descrizione del contenuto della base dati al livello globale. Per questo i report tipicamente contengono dei controlli *calcolati* che forniscono consuntivi, e non permettono la visione di tuple particolari. Un'altra differenza con le maschere è che in genere i report vengono stampati, invece di essere utilizzati in modo interattivo.



**Figura A.18** Una maschera che permette di accedere ad alcuni attributi di PERSONA

## A.5 La definizione di macro

Le macro costituiscono un modo per specificare un insieme di azioni che il sistema deve compiere. In questo modo è possibile automatizzare l'esecuzione di un insieme di compiti. Mediante le macro si possono svolgere le seguenti azioni.

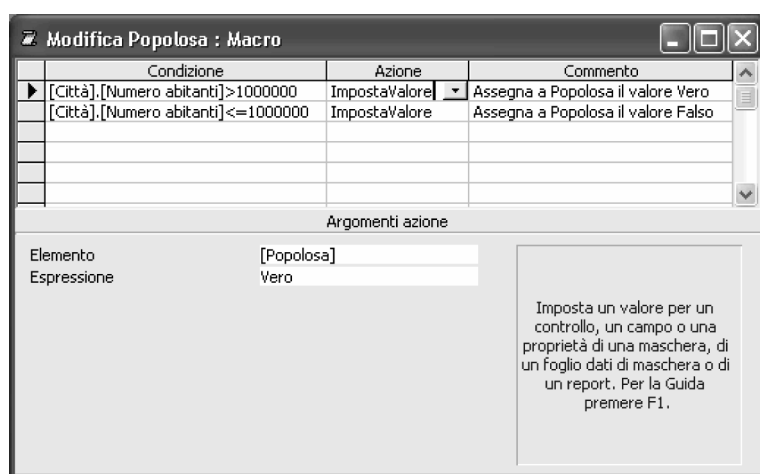
- Far interagire una maschera con altre maschere e con i report. Per esempio, avendo una maschera che descrive i dati dei clienti e una che descrive gli ordini, è possibile aggiungere alla prima maschera un pulsante che attiva la seconda maschera visualizzando solo i dati degli ordini del cliente visualizzato. Si può anche aggiungere un pulsante che permette di stampare un report che elenca la situazione contabile nei confronti del cliente, con l'ammontare di merce rispettivamente ordinata, consegnata e pagata.
- Selezionare e raggruppare automaticamente le tuple. In una maschera si può inserire un pulsante che permette di selezionare immediatamente le tuple che rispettano particolari condizioni.
- Assegnare i valori agli attributi. Usando una macro, è possibile assegnare a un campo di una maschera un valore ottenuto da altri campi o da altre tabelle della base di dati.
- Garantire l'accuratezza dei dati. Le macro sono molto utili per manipolare e validare i dati sulle maschere. Per esempio, si può definire una macro che reagisce a diversi valori di un attributo con diversi messaggi, e garantire in modo sofisticato che i dati inseriti siano corretti.
- Impostare le proprietà delle maschere, dei report e dei campi. Con le macro si possono automatizzare i cambiamenti di qualsiasi proprietà di questi oggetti. Per esempio, è possibile rendere invisibile una maschera quando serve il suo contenuto ma non serve che questo venga visualizzato.
- Automatizzare i trasferimenti dei dati. Se si devono trasferire ripetutamente dati tra Access e altre applicazioni (sia in lettura sia in scrittura), si può automatizzare il compito.
- Creare un proprio ambiente di lavoro. Si può specificare una macro che apre tutto un insieme di tabelle, query, maschere e report tutte le volte che si apre un database, personalizzando eventualmente le barre degli strumenti.
- Specificare le reazioni a certi eventi. Per ogni campo di una maschera è possibile specificare quale macro debba essere eseguita in corrispondenza di ogni evento di accesso, selezione o modifica. Questa caratteristica costituisce la base per la definizione di comportamenti reattivi in Access, che però presentano diverse limitazioni rispetto a quanto è offerto dai trigger; infatti, tali comportamenti scattano in Access solo quando viene utilizzata una maschera particolare per la manipolazione della base di dati, mentre non scattano se la stessa operazione è eseguita direttamente in SQL o tramite un'altra maschera.

Per la definizione di macro bisogna selezionare l'opzione "Crea" e poi attivare l'icona con etichetta Macro. La finestra di progettazione di macro contiene nella metà superiore una tabella con due colonne: Azione e Commento. La macro è composta da una sequenza di azioni descritte nella colonna Azione, ciascuna su una diversa riga, cui può essere associata una breve descrizione. Nella metà inferiore della tabella compaiono un insieme di attributi che, per ogni singola azione, specificano i parametri dell'azione. È possibile specificare delle condizioni che devono essere verificate affinché un

comando venga eseguito, aggiungendo la colonna Condizione tramite l'opzione presente nella toolbar. Si può inoltre far uso di semplici strutture di controllo.

I comandi disponibili possono essere divisi in varie tipologie. Una prima tipologia è costituita dai comandi che permettono di chiedere l'esecuzione di altri servizi, che possono essere altre macro, query, generici comandi SQL, o anche applicazioni esterne. Un'altra tipologia sono i comandi che permettono di accedere ai dati, scandendo il contenuto di una tabella o di una maschera. Vi sono poi i comandi che permettono di manipolare il contenuto della base di dati. Infine, vi sono famiglie di comandi che permettono di trasferire dati tra Access e altre applicazioni, di modificare la dimensione delle finestre e di aprire delle finestre di dialogo con l'utente.

Una semplice macro è descritta in Figura A.19. La macro è associata alle modifiche sull'attributo Numero abitanti di una città; la macro assegna all'attributo Popolosa il valore Sì se il numero di abitanti è superiore a un milione, altrimenti assegna all'attributo il valore No.



**Figura A.19** La finestra di definizione delle macro

# Appendice B

## Il sistema DB2

DB2 appartiene a una storica famiglia di sistemi di gestione di basi di dati prodotti dalla IBM. Il capostipite di questa famiglia è SQL/DS, uno dei primi prodotti commerciali basati sul modello relazionale, reso disponibile dall'IBM agli inizi degli anni Ottanta. A sua volta, il sistema SQL/DS affonda le sue radici in System R, uno dei primi prototipi di DBMS relazionale sviluppato, negli anni Settanta, nei laboratori di ricerca dell'IBM di San Josè. Proprio nell'ambito dello sviluppo di questo prototipo è nato il linguaggio SQL che è poi presto diventato il linguaggio di riferimento per tutti i DBMS commerciali basati sul modello relazionale.

DB2 estende il modello relazionale con un ricco pacchetto di funzionalità avanzate, tra cui:

- un supporto alla gestione di dati multimediali quali testi, immagini, audio e video;
- funzionalità evolute di *business intelligence* per il supporto alle decisioni basate sulla costruzione di *data warehouse* e su operazioni OLAP;
- la possibilità di gestire dati secondo il paradigma a oggetti e dati semistrutturati basati sul linguaggio XML;
- un supporto completo allo sviluppo di applicazioni distribuite e basate su Web che rende possibile l'accesso a basi di dati attraverso la rete Internet;
- un supporto a parallelismi a memorie separate (shared-nothing), nei quali una base di dati viene partizionata tra diversi calcolatori connessi in rete (MMP).

La componente server di DB2 è disponibile su varie piattaforme software e hardware, in ambienti Windows e Unix (nelle loro varie versioni). Per l'accesso al sistema è disponibile una componente client multi-piattaforma scritta in Java. Client remoti possono comunicare con la componente server mediante i più diffusi protocolli di comunicazione (TCP/IP, NetBios, ecc.).

Con DB2 è possibile costruire componenti di federazioni distribuite ed eterogenee di basi di dati, usando un protocollo chiamato *Distributed Relational Database Architecture* (DRDA), adottato da diversi DBMS relazionali (le architetture distribuite sono anch'esse trattate nel secondo volume). Inoltre, DB2 fornisce un supporto per i principali standard di interfacce (quali ODBC, JDBC) e aderisce allo standard corrente di SQL.

Nel resto di questa appendice descriveremo le caratteristiche generali del sistema, prestando particolare attenzione agli strumenti di base e ad alcune sue funzionalità avanzate. Per ulteriori approfondimenti su questo sistema si rimanda il lettore al testo di Chamberlin [1], uno degli inventori di SQL.

## B.1 Caratteristiche generali di DB2

### B.1.1 Versioni del sistema

IBM offre un ricco ventaglio di prodotti per la gestione di dati tra cui un pacchetto per lo sviluppo di data warehouse e un pacchetto per il *content management*, ovvero per la gestione di vari tipi di contenuto digitale. Nel seguito ci occuperemo solo delle funzionalità di *database server* offerte da IBM, ovvero del sistema che si occupa strettamente della gestione di basi di dati, e sebbene venga offerto in varie forme sotto varie denominazioni, utilizzeremo d'ora in poi per la sigla DB2 per fare riferimento a questa componente.

DB2 è disponibile in tre versioni principali, in grado di gestire architetture di complessità crescente.

- *Express*: è una versione di facile installazione e semplificata rispetto alle altre versioni, che offre funzionalità di amministrazione automatica; è adatta allo sviluppo di applicazioni persistenti che richiedano una amministrazione minima della base di dati. Una configurazione entry-level di questa versione chiamata *Express-C* è distribuita gratuitamente da IBM.
- *Workgroup* (WSE): consente l'accesso condiviso a basi di dati locali da parte di utenti/applicazioni locali e remoti ed è disponibile in diversi ambienti. Offre anche funzionalità avanzate di gestione di dati semistrutturati e di accesso alla base di dati tramite dispositivi mobili.
- *Enterprise*: è la versione completa del sistema che, oltre alle funzionalità le altre versioni, consente la gestione di federazioni di basi di dati e il partizionamento di una base di dati tra diversi calcolatori connessi da una rete di comunicazione, ognuno dei quali può essere dotato di più processori.

Il sistema è infine completamente integrato con i diversi ambienti di sviluppo commercializzati da IBM (Eclipse, VisualAge, WebSphere e Rational) che, insieme a DB2, offrono un supporto per l'intero ciclo di vita di applicazioni software realizzate con i più diffusi linguaggi di programmazione.

### B.1.2 Istanze e schemi di DB2

Su un medesimo calcolatore è possibile definire diverse *istanze*<sup>1</sup> indipendenti di server DB2, a ognuna delle quali viene assegnato un nome. Ogni istanza ha una propria configurazione e può gestire diverse basi di dati che restano di proprietà dell'istanza. È possibile in questa maniera adattare il sistema a specifiche necessità applicative; per esempio, si può definire e configurare un'istanza DB2 per applicazioni operative e un'altra, con parametri di configurazione diversi, per applicazioni di supporto alle decisioni. Esiste comunque un'istanza predefinita, creata durante l'installazione di DB2, che si chiama semplicemente DB2. Le basi di dati di una istanza sono organizzate in *schemi* aventi un nome e costituiti da collezioni di tabelle. Lo schema di default nel quale vengono inserite nuove tabelle ha lo stesso nome dell'amministratore DB2.

I client DB2 possiedono una lista delle istanze e delle basi di dati DB2 alle quali possono accedere e sono dotati di strumenti per creare nuove istanze, nuovi schemi

<sup>1</sup> Il termine "istanza" qui usato denota una installazione di DB2 e quindi non ha niente a che vedere con il concetto di istanza di base di dati introdotto nel Libro

e nuove basi di dati. Per interagire con il server, essi devono prima accedere ad una istanza di DB2 e successivamente stabilire una connessione con una base di dati di questa istanza. Un client può connettersi contemporaneamente a diverse basi di dati. Si possono poi inviare comandi DB2 sia a livello di istanza (per esempio per creare una nuova base di dati) che a livello di base di dati (tipicamente una istruzione SQL). Nelle istruzioni SQL si può far riferimento alla tabella di uno schema anteposendo al nome della tabella il nome dello schema, separati da un punto. Se il nome dello schema non è specificato, allora ci si riferisce allo schema di default.

### B.1.3 Interazione con DB2

Come in tutti sistemi di gestione di basi di dati moderni, l'accesso a una base di dati DB2 può avvenire secondo due modalità principali.

- In maniera interattiva, nella quale si inviano, tramite una opportuna interfaccia (comunemente detta *interfaccia utente*), comandi o istruzioni SQL che vengono immediatamente eseguiti dal sistema. Esistono sia una versione semplice dell'interfaccia, puramente testuale, che una versione grafica evoluta e multi-piattaforma scritta in Java. Per l'amministrazione del sistema (creazione di istanze, basi di dati, schemi e gestione di autorizzazioni, prestazioni ecc.) si ricorre nella maggior parte dei casi a questa modalità di interazione.
- Tramite lo sviluppo di programmi in linguaggi di programmazione tradizionali, nei quali vengono immerse istruzioni SQL. È possibile sviluppare, secondo questa modalità, sia programmi statici, nei quali la struttura delle istruzioni SQL è nota a tempo di compilazione, sia programmi dinamici, nei quali le istruzioni SQL vengono generate a tempo di esecuzione.

A differenza di altri sistemi di gestione di basi di dati, DB2 non offre un linguaggio 4GL, cioè un linguaggio di sviluppo ad-hoc. Se da un lato questa scelta comporta la necessità di dover disporre, oltre che di DB2, anche di opportune API per lo sviluppo di applicazioni per basi di dati, dall'altra favorisce la realizzazione di software facilmente portabile da un sistema a un altro.

Nel prossimo paragrafo descriveremo, con maggior dettaglio, come si gestisce una base di dati DB2, secondo le suddette modalità.

## B.2 Gestione di una base di dati con DB2

### B.2.1 Strumenti per la gestione interattiva

La maniera più semplice per interagire con DB2 è attraverso la sua interfaccia utente. Questo avviene tipicamente in una architettura client-server classica nella quale una base di dati che risiede su un server (per esempio su una macchina Unix) viene acceduta dall'utente tramite un client locale (sullo stesso computer sul quale risiede la base di dati) o remoto (per esempio su un personal computer in ambiente Windows). L'interfaccia utente mette a disposizione diversi strumenti interattivi che sono classificati come segue.

- Strumenti di gestione generale (consentono l'amministrazione di una base di dati mediante un'interfaccia grafica di facile uso):

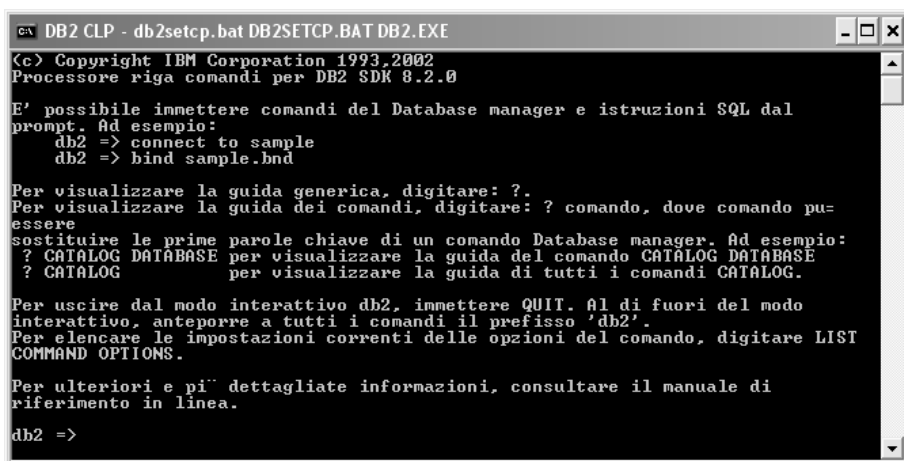
- Il *Centro di controllo* è lo strumento principale e che tipicamente si invoca all'avvio del sistema. Fornisce una interfaccia per le operazioni di amministrazione più importanti quali la creazione di basi di dati e di tabelle. Dal centro di controllo è possibile poi invocare tutti gli altri strumenti.
- Il *Giornale* tiene traccia di tutte le note informative prodotte dal sistema, inclusi i diagnostici e i messaggi di errore. È utile per verificare la presenza di un problema nel sistema.
- Il *Centro attività* consente di pianificare attività che il sistema esegue successivamente in maniera automatica, per esempio dei backup periodici.
- Il *Centro di replica* permette la creazione e la gestione di copie di una base di dati, che vengono tenute aggiornate in maniera automatica dal sistema.
- Strumenti riga comandi (consentono di inviare istruzioni SQL o comandi di sistema).
- Il *Command Line Processor* (CLP) è un ambiente puramente testuale ed è disponibile su tutte le piattaforme.
  - L'*Editor di comandi* è invece dotato di un'interfaccia grafica ed è disponibile per sistemi operativi Windows.
- Strumenti di sviluppo (consentono di realizzare piccole procedure per una basi di dati).
  - Il *Centro di sviluppo* consente la definizione e il testing di oggetti applicativi quali procedure (le stored procedure, descritte nel Libro), tipi utente e funzioni utente (che verranno descritte nei paragrafi B.3.3 e B.3.4).
  - Gli *Strumenti di distribuzione progetto* permettono di esportare su altre basi di dati, eventualmente remote, oggetti applicativi definiti nel Centro di sviluppo.
- Strumenti di informazione:
  - Il *Centro di informazioni* fornisce l'accesso rapido a tutti i manuali DB2.
  - La *Verifica aggiornamenti di DB2* serve a tenere aggiornato il sistema.
- Strumenti di controllo (per il monitoraggio del sistema):
  - Il *Centro di controllo stato* consente di individuare potenziali fonti di errore, per esempio l'eccessivo uso di memoria principale, quando certi indicatori superano soglie di "guardia".
  - L'*Analizzatore di eventi* memorizza tutti gli eventi che si verificano sul sistema e consente di monitorare il corretto funzionamento del sistema.
  - Il *Controllo attività* fornisce informazioni relative alle prestazioni del sistema e all'uso delle risorse.
  - Il *Memory Visualizer* genera grafici relativi all'uso della memoria da parte del sistema.
  - Il *Indoubt Transaction Manager* consente di analizzare transazioni a due fasi che si trovano in stati di incertezza (indoubt), ovvero transazioni preparate ma per le quali non è stato effettuato il commit o il rollback (le transazioni sono descritte nel Libro).
- Strumenti di configurazione:
  - L'*Assistente di configurazione* consente di configurare facilmente il sistema per la connessione a basi di dati remote.
  - I *Primi passi* che fornisce un tutorial sull'uso del sistema.
  - La *Registrazione Visual Studio Add-in* che permette l'installazione di un plugin in Visual Studio che consente l'invocazione diretta di DB2 da questo ambiente.

- Altri strumenti, di vario tipo.
  - Il *Centro licenze* fornisce informazioni sul tipo di licenza installata sul computer.
  - Il *Centro di gestione satelliti* consente la gestione di satelliti, ovvero di gruppi di diversi server DB2 con configurazioni simili sui quali vengono eseguite le medesime applicazioni.
  - L'*SQL assist* offre un supporto in linea per la scrittura di istruzioni SQL.

Nel seguito vengono illustrati con maggior dettaglio gli ambienti che consentono la gestione di base del sistema.

**Command Line Processor** La versione spartana ma completa dell'interfaccia utente di DB2 è costituita da un ambiente testuale chiamato *Command Line Processor* (CLP) ed è disponibile su tutte le piattaforme. Il CLP si può invocare a livello di sistema operativo con il comando `db2`. Un esempio di interazione con il CLP è riportato in figura B.1. Il prompt `db2 =>` indica che il sistema è pronto a ricevere comandi. Una volta entrati nell'ambiente si deve avviare il sistema con il comando `db2start` e poi si possono inviare, oltre alle istruzioni SQL, una serie di comandi per la gestione completa del sistema. Il comando inviato viene eseguito immediatamente e risultato viene mostrato sul video. Se i comandi sono più lunghi di una riga, bisogna segnalare la continuazione sulla riga successiva con il carattere di *backslash* (`\`). Tutti i comandi possono essere inviati direttamente da sistema operativo premettendo il prefisso `db2`.

**Centro di controllo** Permette di amministrare una base di dati mediante un'interfaccia grafica che offre la possibilità di specificare operazioni di definizione di tabelle e di vincoli di integrità, controllo delle autorizzazioni, backup, ripristini, ecc. Come si può vedere in figura B.2, l'interfaccia del Centro di controllo ha tre finestre principali. In quella di sinistra vengono presentati, secondo una organizzazione gerarchica, gli



```

CA DB2 CLP - db2setcp.bat DB2SETCP.BAT DB2.EXE
<c> Copyright IBM Corporation 1993,2002
Processore riga comandi per DB2 SDK 8.2.0

E' possibile immettere comandi del Database manager e istruzioni SQL dal
prompt. Ad esempio:
db2 => connect to sample
db2 => bind sample.bnd

Per visualizzare la guida generica, digitare: ?.
Per visualizzare la guida dei comandi, digitare: ? comando, dove comando pu-
essere
sostituire le prime parole chiave di un comando Database manager. Ad esempio:
? CATALOG DATABASE per visualizzare la guida del comando CATALOG DATABASE
? CATALOG per visualizzare la guida di tutti i comandi CATALOG.

Per uscire dal modo interattivo db2, immettere QUIT. Al di fuori del modo
interattivo, anteporre a tutti i comandi il prefisso 'db2'.
Per elencare le impostazioni correnti delle opzioni del comando, digitare LIST
COMMAND OPTIONS.

Per ulteriori e pi` dettagliate informazioni, consultare il manuale di
riferimento in linea.

db2 =>
  
```

**Figura B.1** Il Command Line Processor di DB2



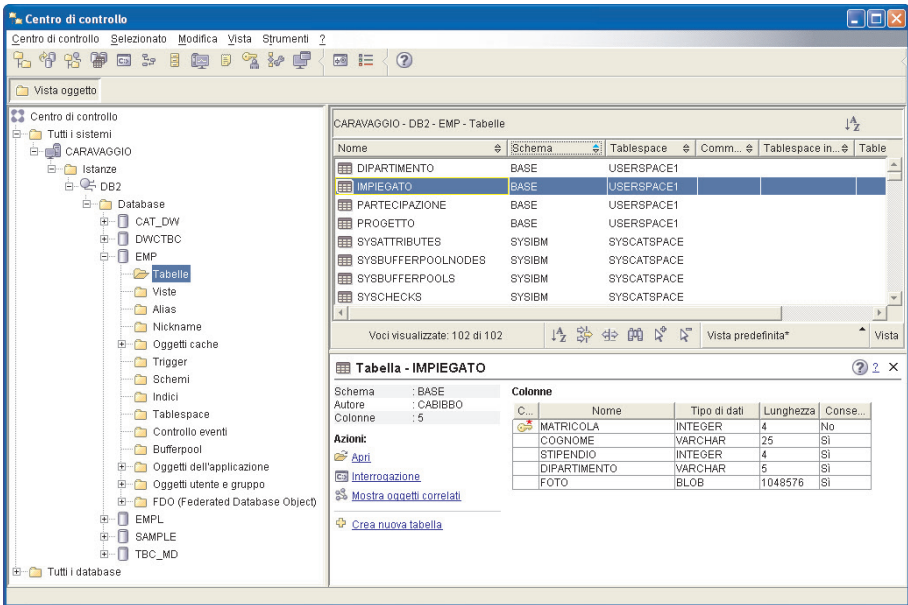


Figura B.2 Il Centro di controllo di DB2

oggetti DB2 (istanze, basi di dati, tabelle, ecc.) che l'utente ha a disposizione. Questi oggetti possono essere dichiarati esplicitamente dall'utente, oppure si può richiedere al Control Center di cercare su rete tutti gli oggetti DB2 ai quali è possibile accedere. Nel nostro caso, si può osservare che è disponibile un sistema con oggetti DB2 chiamato CARAVAGGIO. Le informazioni relative a questo sistema sono state espanse, come indicato dal segno meno davanti alla relativa icona. Su questo sistema risiedono due istanze DB2: una è quella di default, l'altra si chiama MyDB2. La prima contiene diverse basi di dati tra cui una chiamata EMP. Per questa base di dati vengono mostrate tutte le sue componenti, mentre le informazioni relative alle altre non sono state espanse, come indicato dal segno + davanti alle relative icone. La finestra in alto a destra contiene informazioni dell'oggetto selezionato nella finestra di sinistra. Nel nostro caso, sono mostrate tutte le tabelle della base di dati EMP. Questa base di dati contiene due schemi: lo schema BASE e lo schema SYSIBM. Le tabelle dello schema SYSIBM sono tabelle di sistema. Nella finestra in basso a destra vengono infine riportate informazioni sulla tabella selezionata. Nel nostro esempio sono mostrati i dettagli della tabella Impiegato. Se si clicca con il bottone di destra del mouse su un qualunque oggetto sullo schermo, compare un menu che consente di effettuare una serie di azioni sull'oggetto puntato. Per esempio, puntando l'icona Tabelle di una base di dati, è possibile creare una nuova tabella. Il sistema guida l'utente nell'esecuzione di queste operazioni.

**Editor comandi** Questo strumento consente di digitare ed eseguire istruzioni SQL e di comporre *script*, ovvero sequenze di istruzioni SQL da eseguire eventualmente in momenti prestabiliti. Può essere invocato dal Centro di controllo cliccando sull'icona

Interrogazione della finestra in basso a destra. L'Editor comandi è costituito da tre ambienti tra loro integrati: *Comandi*, *Risultati* e *Plan di accesso*; per passare da un ambiente all'altro è sufficiente cliccare sul corrispondente nome. Nel primo è possibile digitare singole istruzioni SQL, script o comandi DB2 di amministrazione. In figura B.3 viene mostrato un esempio di interrogazione SQL formulata nell'Editor comandi di DB2 (invocato dal Centro di controllo).

Si tratta di una interrogazione che cerca gli impiegati che guadagnano più del rispettivo direttore, in uno schema contenente le tabelle:

```
IMPIEGATO (Matricola, Cognome, Dipartimento, Stipendio)
DIPARTIMENTO (Codice, Nome, Sede, Direttore).
```

Per eseguire i comandi digitati è sufficiente cliccare sull'icona di esecuzione, costituita da un triangolo: un messaggio relativo all'esecuzione dell'istruzione (completamento con successo o eventuali diagnostici di errore) viene riportato nella finestra in basso, mentre il risultato viene visualizzato automaticamente nell'ambiente *Risultati dell'interrogazione*. Un esempio del contenuto di questo ambiente dopo aver eseguito l'interrogazione in figura B.3 viene mostrato in figura B.4.

Passando dal primo al secondo ambiente è quindi possibile interagire in maniera interattiva con una base di dati. I comandi (o gli script) digitati nel primo ambiente possono essere salvati su un file per successive esecuzioni. Il terzo ambiente visualizza

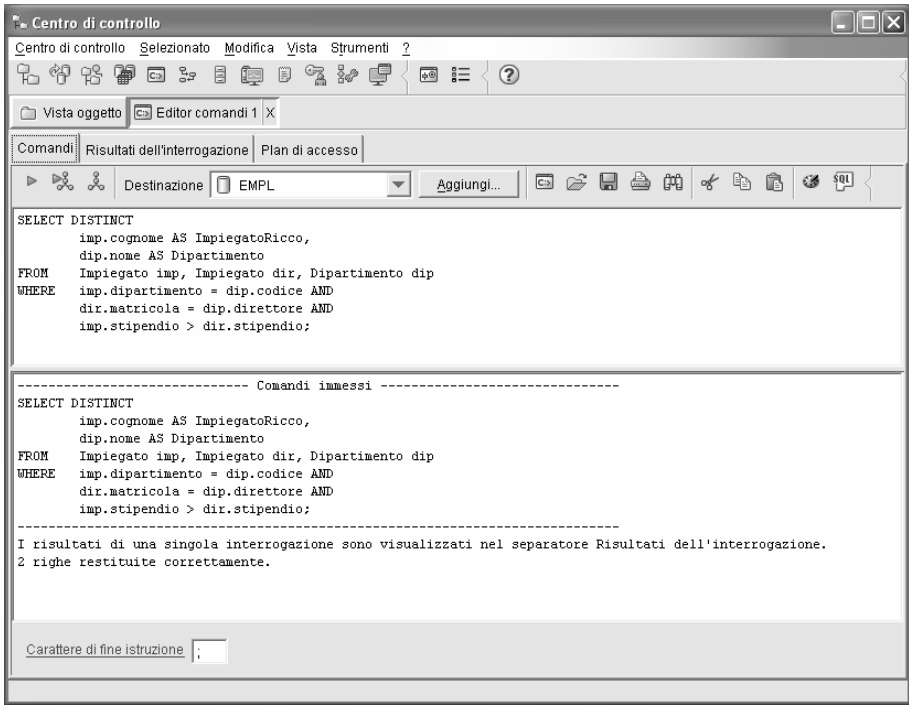


Figura B.3 Una interrogazione SQL fatta con l'Editor comandi di DB2

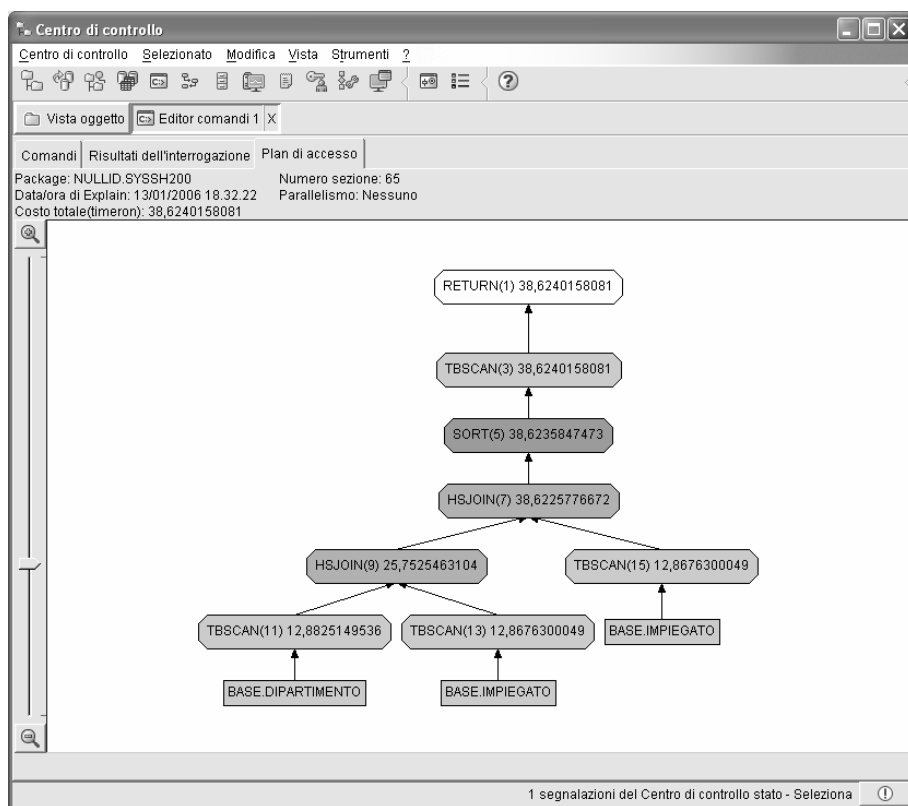


**Figura B.4** Il risultato di una interrogazione SQL nell'Editor comandi di DB2

i piani di accesso creati dall'ottimizzatore di DB2 per l'esecuzione di istruzioni SQL digitate nel primo ambiente. I piani di accesso hanno la forma di alberi: le foglie degli alberi corrispondono alle tabelle coinvolte nell'istruzione SQL e i nodi interi a operazioni effettuate dal sistema; per esempio, la scansione di una tabella, un certo tipo di join tra due tabelle, l'ordinamento di un risultato intermedio. La creazione di questi grafi deve essere richiesta esplicitamente in fase di esecuzione al sistema cliccando sull'icona che raffigura un albero. In figura B.5 viene mostrato una porzione del piano di accesso dell'interrogazione di figura B.3.

Nella finestra sono indicate delle operazioni di preparazione (scan) che precedono le operazioni di join tra le tabelle DIPARTIMENTO ed IMPIEGATO e le operazioni che seguono (sort e scan) per la preparazione del risultato finale. Il valore indicato nei nodi fornisce una stima del costo previsto per l'operazione. Cliccando sui singoli nodi è possibile visualizzare altri dati relativi alla operazione corrispondente, quali stime più dettagliate sui costi di esecuzione e sulla cardinalità dei risultati. L'analisi dei piani di accesso consente di ottimizzare interrogazioni, per esempio introducendo opportunamente degli indici per evitare una operazione di ordinamento durante la sua esecuzione. Questi aspetti sono illustrati in dettaglio nel secondo volume.

**Giornale** Costituisce un utile strumento di verifica perchè mantiene una traccia di tutte le operazioni svolte su una base di dati. In particolare, è possibile visualizzare con questo strumento l'esecuzione di attività pianificate (pannello Cronologia attività), le operazioni di backup/ripristino (pannello Cronologia database), tutti i messaggi generati dal sistema in seguito all'esecuzione delle varie operazioni svolte (pannello Messaggi) e i diagnostici di errore prodotti dal sistema (pannello Registrazione notifiche).



**Figura B.5** Il piano di accesso generato da DB2 per una interrogazione SQL

**Centro di controllo stato** Si tratta di un altro utile strumento che consente di analizzare lo stato del sistema e risolvere preventivamente potenziali problemi. In particolare è in grado di segnalare situazioni “limite”, che non costituiscono ancora veri e propri errori, ma possono indicare possibili malfunzionamenti del sistema. Questi avvisi vengono generati quando alcuni indicatori di stato (per esempio la dimensioni di memoria principale occupata) superano dei valori di soglia predefiniti (detti *threshold*). Un esempio di contenuto del Centro di controllo viene mostrato di figura B.6.

In questo esempio viene segnalato un eccessivo utilizzo della memoria riservata al controllo (92% del valore massimo) e la necessità di eseguire dei backup. Selezionando un avviso, vengono suggerite delle azioni da intraprendere per risolvere la segnalazione.

**Centro informazioni** Fornisce un ricco insieme di informazioni sul sistema DB2. Tramite questo strumento è possibile accedere a spiegazioni su come effettuare certe operazioni (per esempio, “creazione di una tabella” o “backup di una base di dati”), a tutorial, ai vari manuali disponibili in linea, a chiarimenti sulla messaggistica di DB2 e sui codici di errore, e a una lista di programmi di esempio che illustrano le varie funzionalità di DB2.

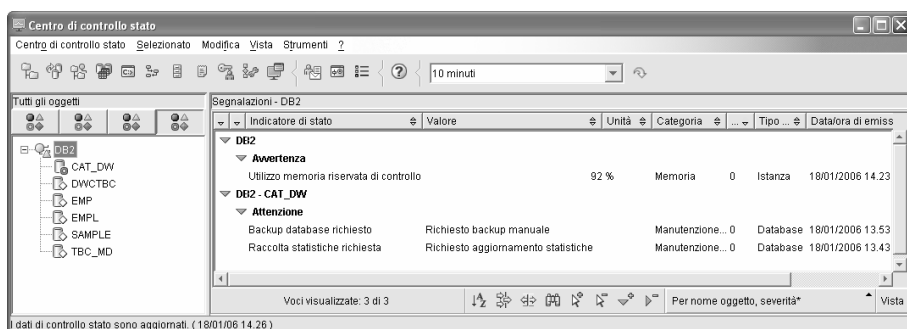


Figura B.6 Il centro di controllo stato di DB2

### B.2.2 Applicazioni

DB2 mette a disposizione diversi strumenti per lo sviluppo di applicazioni per basi di dati, nei quali le istruzioni SQL di interazione con la base di dati vengono usate insieme a istruzioni di linguaggi di programmazione tradizionali, detti linguaggi *host*, come C, C++, Java, FORTRAN e COBOL. Come descritto nel Libro, esistono due modalità principali: l'SQL embedded (statico e dinamico) e l'uso delle Call Level Interface (CLI) quali ODBC e JDBC. Ricordiamo che nell'SQL statico i nomi delle relazioni e degli attributi coinvolti nelle istruzioni SQL sono fissati a priori; l'unica parte delle istruzioni che può rimanere non nota a tempo di compilazione sono gli specifici valori da ricercare o da aggiornare. Nell'SQL dinamico invece, le istruzioni SQL sono generate a tempo di esecuzione; non è quindi necessario specificare relazioni e attributi coinvolti in una istruzione SQL prima della sua esecuzione.

Nel seguito parleremo prevalentemente dell'SQL statico, mentre accenneremo agli strumenti offerti da DB2 per realizzare programmi secondo le altre modalità. Per rendere la trattazione più autocontenuta, vengono ripetute alcune nozioni già presentate nel Libro.

**SQL statico** Un esempio completo di un programma SQL statico immerso in C per DB2 è riportato in figura B.7. Questo programma accede a una base di dati contenente le tabelle IMPIEGATO e DIPARTIMENTO citate nel paragrafo precedente, legge il nome di una località da input e stampa nome e stipendio degli impiegati dei dipartimenti che si trovano in tale località, ordinati per dipartimento.

Si può innanzitutto osservare che il prefisso utilizzato in DB2 per segnalare una istruzione SQL è `exec sql`. Il programma inizia con una serie di direttive di compilazione che includono la libreria di sistema DB2 (chiamata `sqlenv.h`) e il record `sqlca` nel quale vengono memorizzate automaticamente, di volta in volta, informazioni sullo stato del sistema. In particolare, il campo `sqlcode` di `sqlca` contiene un numero intero che codifica il comportamento dell'ultima istruzione SQL eseguita.

Fra le dichiarazioni delle variabili, notiamo che ve ne sono alcune racchiuse in una sezione delimitata dalle parole chiave `declare section`. Queste variabili sono

```
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <sqlenv.h>
exec sql include sqlca;
void main() {
    char CurrDept[10]; /* Dichiarazione variabile programma */
    exec sql begin declare section; /* Dichiarazione variabili host */
        char CognomeImp[10]; /* Cognome Impiegato */
        char NomeDip[10]; /* Nome Dipartimento */
        char SedeDip[15]; /* Sede Dipartimento */
        long StipendioImp; /* Stipendio Impiegato */
        char msgbuffer[500]; /* Buffer per messaggi di errore DB2 */
    exec sql end declare section;
    exec sql declare C1 cursor for /* Dichiarazione cursore */
        select distinct imp.cognome, imp.stipendio, dip.nome
        from impiegato imp, dipartimento dip
        where imp.dipartimento = dip.codice and
              dip.sede = :SedeDip
        order by dip.nome;
    exec sql whenever sqlerror go to PrintError;
    exec sql connect to EMP; /* Connessione alla base di dati EMP */
    printf("Digita una localita':"); scanf("%s", SedeDip);
    exec sql open C1; /* Apre il cursore C1 */
    exec sql fetch C1 into :CognomeImp, :StipendioImp, :NomeDip;
    if (sqlca.sqlcode==100) printf("Nessun dipartimento trovato.\n");
    while (sqlca.sqlcode == 0) {
        if (strcmp(NomeDip,CurrDept)!=0) {
            printf("\nDipartimento:
                %s\nImpiegato\tStipendio\n",NomeDip);
            strcpy(CurrDept, NomeDip);
        } /* end if */
        printf("%s\t%d\n", CognomeImp, StipendioImp);
        exec sql fetch C1 into :CognomeImp, :StipendioImp, :NomeDip;
    } /* end while */
    exec sql close C1; /* Chiude il cursore C1 */
    exec sql connect reset; /* Rilascia la connessione */
    return;
PrintError: /* Trova e stampa un messaggio di errore */
    sqlaintp(msgbuffer, 500, 70, &sqlca);
    printf("Errore DB2 inatteso: %s\n", msgbuffer);
    return;
} /* end of main */
```

**Figura B.7** Un programma SQL immerso in C per DB2

dette *host* e sono quelle variabili del programma che possono essere usate nelle istruzioni SQL: tali variabili realizzano l'interfaccia tra programma e base di dati. Si osservi che quando una variabile *host* è usata in SQL, è preceduta da un prefisso (:); questo prefisso serve a distinguere le variabili dai nomi di attributo. L'istruzione *whenever* permette di stabilire il comportamento del programma al seguito dell'esecuzione di ogni istruzione SQL non andata a buon fine. L'istruzione non è quindi associata a una specifica operazione SQL, ma all'intero programma. Tre opzioni sono possibili

per questa istruzione: `not found`, `sqlerror` e `sqlwarning`. Nel primo caso si può specificare cosa fare quando il risultato di una interrogazione è vuoto (`sqlcode 100`); nel secondo, quando si è verificato un errore (`sqlcode 0`); nell'ultimo, quando si è verificato un *warning*, ovvero la segnalazione di un evento imprevisto ma non grave (`sqlcode > 0`). Nel nostro programma è stato stabilito che, quando si verifica un errore, il controllo passi ad una routine che carica in un buffer il messaggio di errore generato, lo stampa e poi termina il programma. Nulla è stato specificato per risultati nulli e *warning*.

Prima di interagire con una base di dati, va stabilita una connessione con essa tramite l'istruzione `connect to`. Successivamente, si possono eseguire liberamente istruzioni SQL i cui effetti sulla base di dati divengono però permanenti solo dopo l'esecuzione dell'istruzione `commit`. (Il programma di esempio non prevede istruzioni di aggiornamento della base di dati è quindi non fa uso di tale istruzione). Per eseguire le operazioni previste, il programma fa uso della tecnica del cursore presentata nel Libro. Viene inizialmente definito un cursore su una interrogazione SQL che contiene la variabile host `SedeDip`. Successivamente, dopo aver memorizzato in questa variabile un dato letto da input, il cursore viene aperto, l'interrogazione eseguita e, tramite l'istruzione `fetch`, il risultato viene copiato, una tupla alla volta, in variabili di host per poter essere visualizzato. Questa operazione viene coordinata esaminando il valore assunto dalla variabile di stato `sqlcode`. In particolare, viene visualizzato un messaggio quando il risultato è vuoto (`sqlcode` pari a 100) e viene ripetuta l'istruzione di copia e stampa finché ci sono ancora tuple da visualizzare (`sqlcode` pari a 0). Il cursore viene quindi chiuso e la connessione rilasciata.

Per creare una applicazione eseguibile per DB2, il programma va prima precompilato inviando il comando `prep <nome file>` nel CLP o nel Editor comandi. Il risultato della precompilazione va poi compilato e collegato alle librerie usate secondo le modalità adottate per un qualunque programma scritto nel linguaggio host scelto.

**SQL dinamico** Nell'SQL dinamico vengono messe a disposizione speciali istruzioni per compiere le seguenti operazioni principali.

- preparazione di una istruzione SQL con invocazione dell'ottimizzatore DB2 che crea il relativo piano di accesso;
- descrizione del risultato di una istruzione SQL con specifica del numero e del tipo degli attributi;
- esecuzione di una istruzione SQL precedentemente preparata, con assegnamento di valori a eventuali variabili usate nell'istruzione;
- caricamento del risultato, una tupla alla volta, in variabili di programma per il loro successivo uso.

Queste operazioni possono essere immerse in linguaggi di programmazione secondo le medesime modalità dell'SQL statico (in particolare mediante l'uso del prefisso `exec sql`). La fase di preparazione di una istruzione SQL si realizza con il comando `prepare <nome> from <variabile>`, dove la variabile è di tipo stringa e tipicamente memorizza istruzioni SQL. Questa istruzione SQL può contenere dei punti interrogativi che indicano la presenza di parametri che verranno passati in fase di esecuzione. Nel caso di interrogazioni, la gestione del risultato si realizza con il meccanismo del cursore, come avviene con l'SQL statico.

La fase di descrizione si fonda sull'uso di un *descrittore*, ovvero di una struttura di dati che descrive tipo, lunghezza e nome di un numero variabile di attributi del risultato di un'interrogazione. Il sistema DB2 mette a disposizione a questo scopo il descrittore predefinito `sqllda`, che è un record contenente un numero variabile di campi di tipo `sqlvar`, uno per ogni attributo da descrivere. Il numero di questi campi di tipo viene memorizzato nel campo `sqlld` di `sqllda`. I campi `sqlvar` sono a loro volta dei record contenenti, tra l'altro, un campo `sqltype`, che codifica il tipo dell'attributo, un campo `sqllen`, che memorizza la lunghezza dell'attributo, e un campo `sqlname`, che memorizza il nome dell'attributo. Nella fase di preparazione, si può indicare l'uso di questo descrittore con la seguente sintassi: `prepare <nome> into sqllda from <variabile>`. Dopo l'esecuzione di questa istruzione, in `sqllda` viene caricata la descrizione dell'istruzione memorizzata nella variabile specificata. Mediante questa tecnica, è possibile implementare per esempio una interfaccia utente personalizzata, in grado di accettare ed eseguire istruzioni SQL arbitrarie.

Per la fase di esecuzione e caricamento si possono utilizzare le medesime istruzioni viste per l'SQL statico, compreso l'uso di cursori. Accedendo alle informazioni contenute nel descrittore è possibile generare opportunamente la stampa dei risultati di un'interrogazione.

**Call Level Interface** DB2 offre delle interfacce CLI (Call Level Interface) basate sugli standard ODBC (Open DataBase Connectivity) e JDBC (Java DataBase Connectivity), illustrate nel Libro.

Ricordiamo che l'interfaccia basata su ODBC è disponibile per vari linguaggi di programmazione (escluso Java). Tale interfaccia mette a disposizione una serie di funzioni che possono essere direttamente invocate dai programmi per accedere a una base di dati. Per esempio la funzione `sqlconnect()` consente di connettersi a una base di dati DB2, la funzione `sqlprepare()` prepara una istruzione SQL alla sua esecuzione, la funzione `sqlexecute()` esegue una istruzione SQL precedentemente preparata e infine la funzione `sqlfetch()` carica una tupla in variabili host del programma.

L'interfaccia JDBC si basa sullo stesso principio della ODBC, ma è dedicata allo sviluppo di programmi in linguaggio Java ed è quindi orientata agli oggetti. In particolare, questa interfaccia è dotata del metodo `executequery(String)` che prende in ingresso una stringa contenente una istruzione SQL e restituisce un oggetto della classe predefinita `ResultSet`, costituito da un insieme di tuple. Questa classe possiede una serie di metodi che permettono di manipolare l'insieme di tuple contenute negli oggetti della classe. Oltre che applicazioni standard Java, questa interfaccia permette di sviluppare *applets*, ovvero programmi che possono essere caricati ed eseguiti da un browser Web abilitato allo scopo. In questa maniera, basi di dati DB2 possono essere accedute da una qualunque computer connesso a Internet, senza bisogno di installare su di esso la componente client del sistema DB2.

Il grosso vantaggio dei programmi che usano l'interfaccia CLI è che non hanno bisogno di essere precompilati; inoltre, essi possono essere utilizzati su qualunque DBMS che fornisce un supporto per gli standard citati.



## B.3 Funzionalità avanzate

Come accennato all'inizio dell'appendice, DB2 offre tutta una serie di funzionalità avanzate. Pur non essendo tutte standardizzate, queste funzionalità ci danno una interessante indicazione delle caratteristiche dei sistemi di gestione di basi di dati reali. Ne citiamo alcune.

### B.3.1 Dati complessi

DB2 mette a disposizione tre tipi di dati predefiniti che possono essere usati per memorizzare in una tabella dati complessi (per esempio documenti o immagini), che vengono chiamati genericamente LOB (Large Objects).

- *Blob (Binary Large Object)*: rappresenta un dato in formato binario, grande fino a due gigabyte. I dati di tipo Blob non possono essere assegnati o confrontati con dati di altro tipo.
- *Clob (Character Large Object)*: rappresenta un dato composto da una sequenza di caratteri di un byte, grande fino a due gigabyte. I dati di tipo Clob possono essere confrontati con dati di tipo stringa (Char e Varchar).
- *Dbclob (Double-Byte Character Large Object)*: rappresenta un dato composto da una sequenza di caratteri di due byte, grande fino a due gigabyte. I dati di tipo Dbclob possono essere usati solo su basi di dati con una configurazione apposita.

Una possibile definizione di tabella che contiene dati di tipo LOB è la seguente.

```
create table Impiegato (
  Codice integer not null unique,
  Nome varchar(20),
  Stipendio decimal(7,3),
  DataAssunzione date,
  Foto blob(5M) compact,
  Curriculum clob(500K)
)
```

In questa tabella la colonna Foto conterrà immagini e la colonna Curriculum testi. La dimensione specificata tra parentesi indica un valore massimo (si possono usare i suffissi K, M, e G per indicare Kilobyte, Megabyte e Gigabyte). L'opzione `compact` specifica che il dato LOB dovrà occupare il minimo spazio sul disco, a costo di una minore efficienza nella sua gestione.

Esistono diverse limitazioni nell'uso di dati LOB in istruzioni SQL. In particolare non è possibile confrontare direttamente colonne di tipo LOB mediante operatori come `=`, `>`, `<` o `in`. È però possibile far uso dell'operatore `like`. Per esempio, l'istruzione SQL:

```
select Codice, Nome
from Impiegato
where Curriculum like '%DBA%'
```

trova gli impiegati per i quali compare la stringa 'DBA' nel curriculum.

I dati LOB sono gestiti dal sistema in maniera da minimizzare i loro spostamenti da una locazione di memoria a un'altra. In particolare, i LOB possono essere manipolati nelle applicazioni per mezzo di opportune variabili dette *locatori*. I locatori rappresentano un LOB senza memorizzarlo fisicamente. Usando in maniera opportuna i locatori è possibile differire o addirittura evitare il caricamento di un LOB in un programma. Per esempio, la copia di un LOB avviene semplicemente copiandone il locatore. Un locatore si definisce nella sezione di dichiarazione delle variabili host di un

programma con il comando: `sql type is <tipo di LOB> <nome locatore>`. Successivamente, il locatore può essere usato come tutte le altre variabili host per gestire un LOB del tipo associato. È inoltre possibile manipolare locatori con funzioni speciali tra cui `posstr`, che trova la posizione della prima occorrenza di un pattern in un LOB, e `substr`, che restituisce la sottostringa di un LOB compresa tra le posizioni specificate.

### B.3.2 Extender

Oltre ai tipi predefiniti di base, DB2 mette a disposizione, come prodotti ausiliari, degli *extender*, ovvero ulteriori tipi di dato, più specifici dei LOB, per la gestione di dati non tradizionali. A ciascun extender sono associate particolari funzioni ausiliarie per la manipolazione dei relativi tipi di dato. Generalmente queste funzioni possono essere utilizzate liberamente in comandi SQL.

Gli extender sono tipicamente sviluppati direttamente dalla IBM ma esistono anche extender sviluppati da partner commerciali della IBM.

Tra gli extenders DB2 più interessanti citiamo i seguenti.

- Gli *AIV (Audio, Image, Video) extenders* per la gestione di dati multimediali. Questi extender offrono funzionalità evolute di manipolazione di dati multimediali (tra cui la riproduzione di suoni e video) e di ricerca avanzata (per esempio ricerche basate su proprietà dei dati quali la presenza di un certo campione audio o immagine).
- Il *text extender* per la gestione di basi di dati documentali. Questo extender offre funzionalità tipiche dell'*information retrieval* come l'indicizzazione di testi e la ricerca basata su parole chiavi.
- Lo *spatial extender* per la gestione di dati spaziali. Vengono offerti a tale riguardo diversi tipi di dato spaziale che possono essere usati per modellare oggetti del mondo reale quali mappe, confini geografici, corsi di fiumi, ecc.
- L'*XML extender* per la gestione di dati in formato XML. Questo extender consente di memorizzare documenti XML in attributi di tabelle relazionali. Le funzioni messe a disposizione permettono di effettuare ricerche su interi documenti XML o su loro porzioni e di trasformare dati XML in basi di dati relazionali e viceversa.
- Il *Net search extender* per la gestione di ricerche testuali evolute. Offre un ricco insieme di funzioni di ricerca su testi e di indicizzazione di documenti memorizzati in una base di dati. Questi strumenti consentono la costruzione di motori di ricerca.

### B.3.3 Tipi utente

Un tipo di dato utente (denominato *distinct* in DB2) è un tipo definito a partire dai tipi di dato di base e corrisponde al concetto di definizione di dominio utente introdotto nel libro. Per esempio, la definizione dei tipi MONEY, IMAGE e TEXT può essere specificata con le seguenti istruzioni:

```
create distinct type Money as decimal(7,2)
with comparisons;
create distinct type Image as blob(100M);
create distinct type Text as clob(500K) compact;
```

La clausola *as* specifica il tipo *sorgente* del tipo utente, mentre la clausola *with* serve a specificare che i relativi dati vanno confrontati come se fossero dati dei rispettivi tipi sorgente. Si possono usare come tipi sorgente solo tipi di dato DB2 predefiniti e non sono ammesse nidificazioni nelle definizioni.

Una volta definiti, i tipi utente possono essere liberamente usati nelle creazioni di tabelle. Per esempio, la precedente definizione della tabella IMPIEGATO può essere riscritta come segue:

```
create table Impiegato (
  Codice integer not null unique,
  Nome varchar(20),
  Stipendio money,
  DataAssunzione date,
  Foto image,
  Curriculum text
)
```

Sugli attributi definiti su un tipo utente non è in generale possibile applicare le medesime operazioni che si possono applicare sui rispettivi tipi sorgente. Per esempio, non è possibile sommare due dati di tipo Money. Si può però ovviare a questa limitazione con la definizione di opportune funzioni utente, come descritto nel seguito.

### B.3.4 Funzioni utente

Le funzioni utente corrispondono alle procedure (o *stored procedure*) introdotte nel libro. In DB2 possono essere dichiarate in maniera esplicita mediante l'istruzione `create function` che assegna un nome alla funzione e ne definisce la semantica. Tali funzioni sono associate a una base di dati e possono essere utilizzate solo nel contesto della rispettiva base di dati. Una caratteristica importante delle funzioni utente DB2 è che aderiscono al principio di *overloading* della programmazione orientata agli oggetti. È possibile definire cioè funzioni con lo stesso nome purché i parametri di ingresso delle varie definizioni differiscano in tipo e/o numero. In base allo stesso principio è anche possibile ridefinire una funzione DB2 predefinita, per esempio gli operatori aritmetici. Le funzioni utente DB2 possono essere classificate come segue.

- *Funzioni interne*: si costruiscono sulla base di funzioni DB2 predefinite, dette funzioni *sorgente*, in maniera simile a quanto avviene per i tipi utente.
- *Funzioni esterne*: corrispondono a programmi esterni alla base di dati e scritti in linguaggi di programmazione di alto livello (C o Java) che possono contenere o meno istruzioni SQL. La dichiarazione di una funzione esterna contiene la specifica della locazione su disco dove risiede il codice che implementa la funzione. Esistono due tipi di funzioni esterne:
  - *Funzioni scalari*: possono ricevere diversi valori in ingresso ma restituiscono un solo valore in uscita; se il nome della funzione ridefinisce un operatore (per esempio “+”) allora tali funzioni possono essere invocate utilizzando la notazione infissa;
  - *Funzioni tabella*: sono in grado di restituire una collezione di ennuple di valori, che può essere assimilata a una tabella: ogni volta che queste funzioni vengono invocate, esse restituiscono una nuova ennupla (vista come una riga di tabella), oppure un codice speciale che indica il completamento dell'operazione.

Le funzioni interne si usano principalmente con i tipi utente per poter applicare, in maniera selettiva, alcune funzioni dei relativi tipi sorgente. Per esempio, con riferimento al tipo utente Money precedentemente definito, possiamo definire alcune funzioni interne con le seguenti dichiarazioni.

```
create function "*" (Money, Decimal()) returns Money
    source "*" (Decimal(), Decimal())
create function Total (Money) returns Money
    source sum (Decimal())
```

In queste dichiarazioni viene definito il nome della funzione, i tipi dei parametri di ingresso (che possono essere anche tipi utente), il tipo del dato restituito e la funzione *sorgente*, ovvero la funzione DB2 che va applicata quando viene invocata la funzione utente che si sta definendo. Nel nostro caso è stato ridefinito l'operatore di prodotto ed è stata definita una nuova funzione basata sull'operatore aggregativo `sum` di SQL. Queste dichiarazioni rendono legali istruzioni SQL come le seguenti.

```
select Dipartimento, Total (Stipendio)
from Impiegato
group by Dipartimento;
update Impiegato
set Stipendio = Stipendio * 1.1
where Dipartimento = 'Produzione';
```

Non sono invece valide espressioni che coinvolgono, per esempio, la somma di uno stipendio e di un numero decimale perchè l'operatore di somma risulta indefinito per il tipo Money.

Supponiamo ora di voler definire una funzione scalare esterna che calcola lo stipendio di riferimento dovuto a un impiegato in base alla sua anzianità di servizio, valutando l'anzianità sulla base della data di assunzione. Una possibile definizione per tale funzione esterna è la seguente:

```
create function StandardSalary (Date) returns Money
    external name '/usr/db2/bin/salary.exe!StdSal'
    deterministic
    no external action
    language c parameter style db2sql
    no sql;
```

questa dichiarazione definisce la funzione esterna `StandardSalary` che riceve una data in ingresso e restituisce un dato di tipo Money. La clausola `external` indica il nome e la posizione su disco del file che contiene il codice della funzione. Il nome dopo il punto esclamativo indica il modulo (funzione C nel nostro caso), all'interno del file, che implementa la funzione utente. La clausola `deterministic` specifica che diverse invocazioni della funzione sullo stesso valore restituiscono sempre lo stesso risultato. La clausola `external action` serve a specificare se la funzione coinvolge azioni esterne alla base di dati, come per esempio la scrittura in un file. Questa informazione può essere utile all'ottimizzatore per decidere se limitare il numero di invocazioni della funzione stessa. La clausola `language` specifica il linguaggio di programmazione usato per implementare la funzione esterna e le modalità di passaggio di parametri con essa.

La modalità standard per il linguaggio C è denominata `db2sql`, mentre quello

per il linguaggio Java è `db2general`. Infine, l'ultima clausola indica che la funzione non coinvolge accessi a una base di dati.

Non discuteremo qui le modalità secondo le quali si implementa una funzione esterna. Accenniamo solo al fatto che esistono delle convenzioni per trasformare tipi di dato SQL in tipi di dato del linguaggio di programmazione prescelto e viceversa, e che l'implementazione ha in effetti più parametri della funzione esterna. Questi parametri aggiuntivi servono per scambiare ulteriori dati di servizio, come eventuali messaggi di errore.

Una volta definita e implementata una funzione esterna, è possibile usarla in istruzioni SQL nella stessa maniera in cui si usano le funzioni predefinite. Per esempio, la seguente istruzione trova cognomi e stipendi degli impiegati che guadagnano più del 20% del rispettivo stipendio di riferimento, mostrando anche tale valore.

```
select Nome, Stipendio, StandardSalary(DataAssunzione)
from Impiegato
where Stipendio > StandardSalary(DataAssunzione) * 1.2;
```

Si osservi che la condizione nella clausola `where` è valida perché: (a) la funzione utente restituisce un dato di tipo `Money`, (b) il prodotto tra questo tipo di dato e un decimale è stato definito, (c) il valore restituito dal prodotto è di tipo `Money` e può quindi essere confrontato con l'attributo `Stipendio`, come stabilito all'atto della definizione del tipo utente `Money`.

Le funzioni tabella costituiscono una funzionalità particolarmente interessante di DB2. Permettono infatti di trasformare facilmente una qualunque sorgente di dati esterna in una tabella manipolabile con SQL. L'unica cosa da fare è scrivere un programma che accede alla sorgente, per esempio un file di testo o un file MS Excel, filtra eventualmente dei dati in base a parametri passati in ingresso, e infine li restituisce, una riga alla volta. La restituzione delle varie righe avviene mediante una tecnica particolare: viene allocata in maniera automatica una zona di memoria (detta *scratchpad*) che preserva il suo contenuto tra una invocazione e un'altra di una funzione. In questa maniera, una invocazione può accedere a informazioni relative all'invocazione precedente; per esempio, l'ultima posizione del file acceduta.

Supponiamo di avere a disposizione una funzione di questo tipo che prende il nome di una località in ingresso e restituisce una ennupla di valori memorizzati su un file remoto, corrispondenti a dati di vendita di negozi che si trovano in tale località. La definizione della corrispondente funzione tabella potrebbe essere la seguente:

```
create function Vendite(Char(20))
  returns table (Negozio char(20),
               Prodotto char(20),
               Incasso Integer)
  external name '/usr/db2/bin/sales'
  deterministic
  no external action
  language c parameter style db2sql no sql
  scratchpad
  final call disallow parallel;
```

Si può notare che, a parte alcune clausole specifiche per le funzioni tabella, la definizione è simile alla definizione di funzioni scalari esterne. La differenza principale è che

il risultato della funzione è composto da più parametri, che vengono interpretati come attributi di una tabella.

Una possibile istruzione SQL che fa uso della funzione tabella sopra definita è la seguente:

```
select Negozio, SUM(Incasso)
from table(Vendite('Roma')) as VenditeRoma
where Product = 'Giocattoli'
group by Negozio;
```

Questa istruzione restituisce l'incasso totale dei negozi di Roma relativo alle vendite di giocattoli. La tabella interrogata non è memorizzata nella base di dati (come indicato dalla parola chiave `table`), ma viene generata dalla funzione tabella `Vendite`, alla quale viene passato, come parametro di ingresso la stringa `'Roma'`.

## Bibliografia

[1] D. D. Chamberlin. *A Complete Guide to DB2 Universal Database*. Morgan Kaufmann, San Francisco, Calif., 1998.



## Appendice C

### DBMS open source: Postgres

Lo sviluppo del mondo dell'*open source* rappresenta sicuramente uno dei fenomeni recenti di maggiore impatto nel mondo dell'informatica. Sfruttando le caratteristiche di facile riproduzione e trasmissione grazie a Internet dei programmi e lo spirito di partecipazione volontaria di molti esperti di tecnologia informatica, questo movimento ha portato allo sviluppo di un ampio insieme di componenti software e applicazioni con codice sorgente liberamente disponibile. Il frutto più significativo di questo movimento è sicuramente il sistema operativo Linux, il quale, anche beneficiando di iniziative preesistenti come l'ambiente GNU sviluppato sotto la spinta della Free Software Foundation, ha fornito una piattaforma completa e tecnicamente valida che ha dato un impulso fortissimo allo sviluppo di nuovi progetti. Le iniziative del mondo open source si sono sviluppate su numerosissimi fronti, con la realizzazione di browser come Firefox, client di posta elettronica come Thunderbird, server Web come Apache Server, ambienti completi di automazione di ufficio come OpenOffice/LibreOffice. In tutti questi casi, i prodotti open source costituiscono delle valide alternative rispetto alle offerte dei produttori commerciali, con un seguito significativo e in continua crescita nella comunità degli utenti, producendo un impatto che travalica il dominio del sistema operativo Linux.

Il movimento open source ha coinvolto anche il mondo dei DBMS, si potrebbe dire *inevitabilmente* dato il ruolo centrale che i DBMS relazionali rivestono nell'architettura dei moderni sistemi informativi. Questo interesse ha dato luogo a diverse iniziative, spesso in concorrenza tra di loro. In parte come risposta a queste iniziative, si è anche assistito recentemente al rilascio come strumenti gratuiti di versioni limitate dei prodotti commerciali di maggiore diffusione. Inoltre sia IBM, sia Microsoft, sia Oracle, ossia ciascuno dei tre grandi produttori commerciali di DBMS attualmente presenti sul mercato, coordinano iniziative che consentono agli studenti di corsi universitari di disporre di versioni complete dei motori relazionali da loro prodotti a costo gratuito, spesso a condizione che sia attivata una particolare convenzione con l'università, con la limitazione che l'uso del sistema sia a fini educativi.

Vi sono poi prodotti che per varie ragioni, dopo essere stati sviluppati inizialmente come prodotti commerciali, sono poi evoluti in prodotti open source; per esempio, da Interbase è derivato Firebird, da Cloudscape deriva Apache Derby. I database server open source che vedono attualmente maggior seguito sono MySQL e Postgres.



MySQL è ancora oggi il prodotto relazionale che presenta il maggior numero di adozioni all'interno della comunità open source. Si tratta di un prodotto che nasce con l'ambizione di offrire un supporto per la gestione relazionale di dati nel contesto di applicazioni che non generano scritture concorrenti, presentando come punto di forza la facilità di installazione e configurazione e le ottime prestazioni nell'ambito degli scenari applicativi per cui era pensato, tra i quali figura con particolare importanza la costruzione di siti Web. Dato il successo incontrato, gli sviluppatori hanno di recente cercato di far evolvere il sistema con l'ambizione di offrire l'insieme di servizi che caratterizza tipicamente un DBMS relazionale, ma l'iniziativa ha incontrato diversi ostacoli, facilmente prevedibili dato il ruolo cruciale che riveste all'interno di un sistema relazionale la gestione di accessi concorrenti e il supporto per le transazioni.

PostgreSQL rappresenta invece la moderna incarnazione del sistema Postgres (questo è il nome che continua a essere adottato dalla comunità ed è il nome che useremo in questo testo), un progetto nato nella seconda metà degli anni '80 presso l'Università della California a Berkeley, sotto la supervisione di Mike Stonebraker. Il sistema era nato con l'obiettivo di indagare la frontiera della tecnologia delle basi di dati, costruendo un sistema in grado di offrire supporto a diverse innovazioni che sono state studiate negli anni. Al di là di questi aspetti innovativi, il sistema presentava un supporto moderno per l'insieme dei servizi che caratterizza un pieno DBMS relazionale, riconoscendo un insieme ampio di costrutti SQL e gestendo i vari aspetti legati al supporto transazionale.

Nell'ambito della comunità open source Postgres è stato spesso penalizzato da una reputazione che lo considerava un sistema lento e pesante, rispetto alla struttura leggera ed efficiente di MySQL. Una parte di questa reputazione era legata ad alcune scelte di progetto che imponevano una certa rigidità e lentezza al sistema, in parte al necessario onere computazionale e conseguente rallentamento imposto dalla corretta gestione dei requisiti transazionali. Versioni recenti dimostrano un livello di prestazioni estremamente competitivo con quello offerto dalle versioni di MySQL che offrono il supporto transazionale. Data la maggiore flessibilità e il supporto naturale per tutti i servizi che caratterizzano un vero DBMS, è stato naturale nel contesto di questo libro scegliere Postgres come sistema di riferimento per illustrare le caratteristiche di un DBMS open source. Un'ulteriore testimonianza della qualità di Postgres è il fatto che sia stato scelto come DBMS di riferimento per il sistema operativo Mac OS X Server.

## C.1 Caratteristiche del sistema

La versione corrente di Postgres è sempre disponibile presso il sito del progetto <http://www.postgresql.org>. Il progetto è coperto da una licenza open-source di tipo BSD, ovvero una licenza "non virale", meno restrittiva della licenza GPL che caratterizza molti prodotti open-source. Il principale vincolo posto dalla licenza è che, nel momento nel quale si intende utilizzare il sistema all'interno di un proprio prodotto, si deve dichiarare che il sistema ha al suo interno componenti che sono protetti da quella

licenza. Vi è quindi una significativa possibilità di utilizzare il sistema anche all'interno di iniziative commerciali.

Il nucleo principale di Postgres è rappresentato dal motore relazionale, il server che gestisce le tabelle e accetta query dagli utenti del sistema. Il server è multiplatforma e può essere eseguito sia in ambiente Linux, sia Windows, sia Mac OS X. Il server è nato inizialmente per sistemi Unix e nell'uso mostra chiaramente questa origine. In ambito Windows il sistema offre un modello di installazione, gestione e interazione che è coerente con quello delle applicazioni Windows native, ma appena si cercano di utilizzare funzionalità avanzate, la natura originale del sistema si rivela chiaramente. Dato l'obiettivo del sistema e il target di utenti cui il sistema tipicamente offre le proprie interfacce (gestori di sistema e programmatori di applicazioni), questa eterogeneità rispetto al consueto modello di interazione di Windows costituisce una limitazione abbastanza lieve. Data la migliore coerenza con il mondo Unix, tenendo inoltre conto che il mondo delle basi di dati per Windows è trattato nell'appendice dedicata a Microsoft Access, si è scelto di far riferimento in questa appendice a un'installazione su ambiente Linux. In particolare, le sessioni sono state realizzate su una distribuzione Fedora. L'ambiente Ubuntu, oggi la distribuzione Linux più diffusa, presenta minime differenze nella procedura d'installazione e offre gli stessi servizi.

Lo scopo di questo capitolo non è ovviamente quello di fornire una guida dettagliata all'uso di Postgres. A questo fine, esiste una documentazione curata e di ampie dimensioni; per esempio, il manuale standard offerto assieme alla distribuzione consiste al momento di più di 2000 pagine. L'obiettivo è bensì quello di illustrare sinteticamente quali sono le caratteristiche di base del sistema e le funzionalità offerte, per facilitare un'esperienza diretta d'uso che, comunque, avrà bisogno in molti casi di far riferimento alla documentazione originale per risolvere gli aspetti di dettaglio relativi alla configurazione e sviluppo di un'applicazione.

## C.2 Installazione e prima configurazione

Mostriamo i passi necessari per installare il sistema. La maggior parte delle moderne distribuzioni Linux mette già a disposizione un'installazione completa di Postgres. Qualora questa non sia presente o si desideri aggiungere qualche componente, sarà necessario procedere all'installazione manualmente. L'installazione può fare uso degli strumenti di configurazione della distribuzione (`yum` per Fedora, invocando come *root* da shell il comando `yum install postgresql`), o può utilizzare strumenti di livello più basso quali `rpm`, o può addirittura partire dai sorgenti C del sistema compilandoli direttamente. Un vantaggio degli strumenti di più alto livello è la possibilità di mantenere aggiornato in modo facile il sistema in seguito al rilascio di nuove versioni, oltre a disporre di un meccanismo di verifica delle dipendenze tra i diversi componenti molto più robusto. Per quanto riguarda la fonte delle versioni compilate, ci si può appoggiare alla versione rilasciata dal gestore della distribuzione, o si può modificare la propria configurazione esplicitando come sorgente per Postgres il sito ufficiale del sistema, disponendo quindi di versioni più recenti e con un maggior numero di componenti a scelta, a scapito del rischio di incompatibilità con alcuni componenti standard della propria distribuzione Linux. Per seguire questa strada in Fedora è necessario intervenire sulla configurazione locale di `yum`.

Nella configurazione standard, l'installazione del database crea un account di nome `postgres` sul sistema. Il primo passo nell'uso del sistema consiste nell'usare questo account (via `su` o `sudo`) per inizializzare un database vuoto tramite la coppia di comandi `initdb` e `createdb`, di cui sarà proprietario l'utente `postgres`. Le norme di buona gestione di un sistema richiedono di fare attenzione a utilizzare account con troppi privilegi e di cercare di configurare i sistemi secondo il classico principio del "least privilege". In questa trattazione l'attenzione è rivolta all'apprendimento dei principi e funzionalità del sistema, piuttosto che alla installazione e configurazione di un sistema per un ambiente di produzione. Chiaramente nella realizzazione di un'applicazione reale, particolarmente se esposta verso Internet, è necessario adottare tutte le buone pratiche di costruzione di sistemi sicuri, dedicando una particolare attenzione alla corretta configurazione degli aspetti di sicurezza. Il modo più semplice per gestire la protezione consiste nel creare dall'account `postgres` un nuovo ruolo/utente nel database e poi concedere al nuovo utente la possibilità di costruire con il comando `createdb` una propria base di dati, di cui l'utente sarà proprietario; in questo modo si protegge l'integrità del sistema, che è invece esposta a rischi maggiori qualora si operi utilizzando direttamente l'account `postgres`. La creazione del nuovo ruolo/utente può essere ottenuta invocando `create role` all'interno dell'interprete SQL o il comando `createuser` dalla shell.

Una volta che il database server è stato inizializzato, il motore relazionale può essere avviato; per esempio in Fedora si può attivare il server con il comando `service postgresql start`. Il sistema è quindi pronto per accettare le richieste degli utenti autorizzati.

### C.2.1 Interazione testuale: `psql`

Una volta installato, il motore relazionale risponde a richieste che giungono dalle applicazioni aprendo un canale di comunicazione su una porta TCP. Oltre a questa modalità, il sistema consente anche di utilizzare una serie di strumenti che permettono di interagire direttamente con il sistema. Uno strumento di interazione molto utilizzato con Postgres è lo strumento `psql`, un programma che gestisce un'interfaccia di dialogo testuale, con un accesso diretto verso l'interprete SQL del motore relazionale. Per accedere al database di default, è sufficiente invocare `psql` dalla shell. Nella configurazione standard di Postgres è possibile accedere al DBMS senza fornire una password, basandosi su una corrispondenza diretta tra gli account del sistema operativo e gli account del DBMS. Si noti che i due sistemi sono indipendenti, ovvero gli account del sistema operativo non corrispondono agli account gestiti sul database server, ed è relativamente facile modificare la configurazione del sistema per far sì che a ogni accesso al DBMS sia richiesta una verifica esplicita della identità dell'utente.

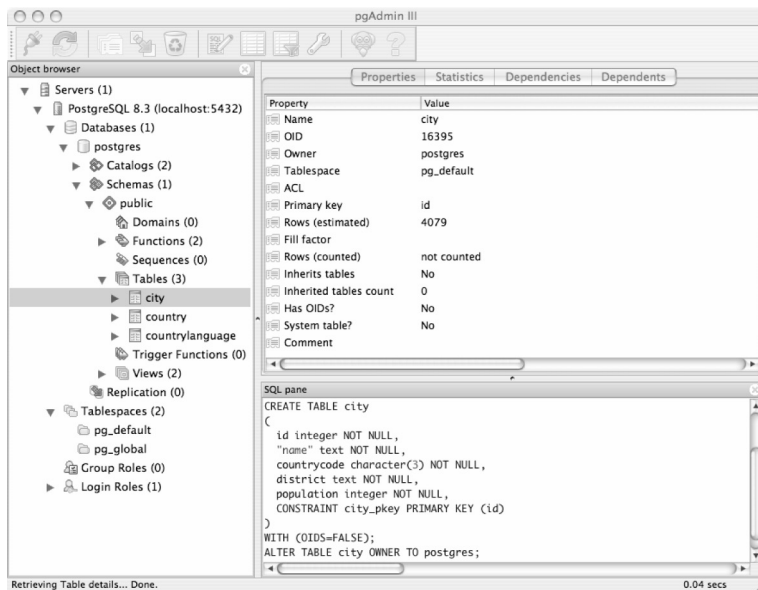
In seguito all'invocazione del comando `psql`, il sistema presenta questa interfaccia:

```
Welcome to psql 8.3.5, the PostgreSQL interactive terminal.
Type: \copyright for distribution terms
      \h for help with SQL commands
      \? for help with psql commands
      \g or terminate with semicolon to execute query
      \q to quit
postgres=#
```

Il sistema è quindi pronto per accettare le richieste dell'utente. Il comando `psql` può anche essere usato in altro modo, fornendo come parametro d'invocazione il nome di un file, tramite l'opzione `-f` oppure tramite il meccanismo di ridirezione dello standard input gestito dalla shell. Per esempio, si può invocare dalla shell Linux il comando `"psql -f ./script.sql"`, equivalente a `"psql < ./script.sql"`. Il file `script.sql` conterrà di norma il testo di una sequenza di comandi SQL, che verranno eseguiti in ordine come se venissero immessi direttamente dall'utente al prompt. Al termine dell'esecuzione della sequenza di comandi, `psql` termina.

## C.2.2 Interazione grafica: pgAdmin

La distribuzione Postgres presenta anche il componente pgAdmin, ora giunto alla terza versione, il quale offre un'interfaccia grafica in grado di gestire in modo sofisticato i compiti di gestione della base di dati. In ambiente Linux, il componente viene invocato dal comando `pgadmin3`; qualora non sia presente, l'installazione in Fedora può avvenire invocando come root il comando `yum install pgadmin` (a condizione che `yum` sia configurato per recuperare i componenti della distribuzione completa di Postgres). Grazie a questa applicazione, anche Postgres dispone di un'interfaccia grafica di gestione, in modo coerente con l'evoluzione degli ultimi tempi di tutti i DBMS commerciali, i quali, a partire dai sistemi Microsoft, hanno negli ultimi anni spostato sempre più l'attenzione verso l'uso di interfacce di questo tipo, riducendo pian piano il ruolo delle classiche interfacce testuali. La schermata iniziale è mostrata nella Figura C.1.



**Figura C.1** La finestra di partenza di pgAdmin

Come avviene in molti di questi ambienti, l'interfaccia è strutturata in un insieme di pannelli.

- *Object browser*: sul lato sinistro in figura, presenta in un formato ad albero navigabile la struttura di componenti del sistema. A ogni componente è associato un menu contestuale, attivabile con il tasto destro del mouse, che mette a disposizione i consueti comandi che consentono di operare sull'oggetto (creando per esempio dei sotto componenti).
- *Information pane*: in alto a destra nella figura, presenta le proprietà dell'oggetto selezionato nell'Object browser. Questo pannello offre diverse prospettive sull'oggetto, rappresentate dai diversi tab: *Properties* descrive le proprietà dell'oggetto, recuperandole dal catalogo della base di dati; *Statistics* descrive dati sintetici relativi all'istanza dell'oggetto selezionato (i cosiddetti profili, che sono discussi nel capitolo 11); *Dependencies* e *Dependents* descrivono i componenti dello schema che sono coinvolti in dipendenze con l'oggetto selezionato.
- *SQL pane*: in basso a destra nella figura, presenta in formato SQL le caratteristiche dell'oggetto selezionato. Il codice SQL presentato viene ricostruito a partire dal contenuto del catalogo; non coincide quindi necessariamente con il codice SQL che è stato utilizzato per generare l'oggetto, ma ne rappresenta una versione in grado di produrre per il motore relazionale esattamente lo stesso componente nello schema.

Lo strumento presenta poi un insieme abbastanza esteso di comandi, disponibili sia come bottoni sulla tool bar (nella figura vi sono 11 bottoni, appena sotto la barra dei menu), sia come voci all'interno dei vari menu. L'insieme è ricco e consente di gestire buona parte delle necessità di interazione con il DBMS. Diversi dei bottoni presenti nella tool bar aprono finestre dedicate. Per esempio, premendo il bottone etichettato "SQL" si apre una finestra per la gestione di query, rappresentata in Figura C.2.

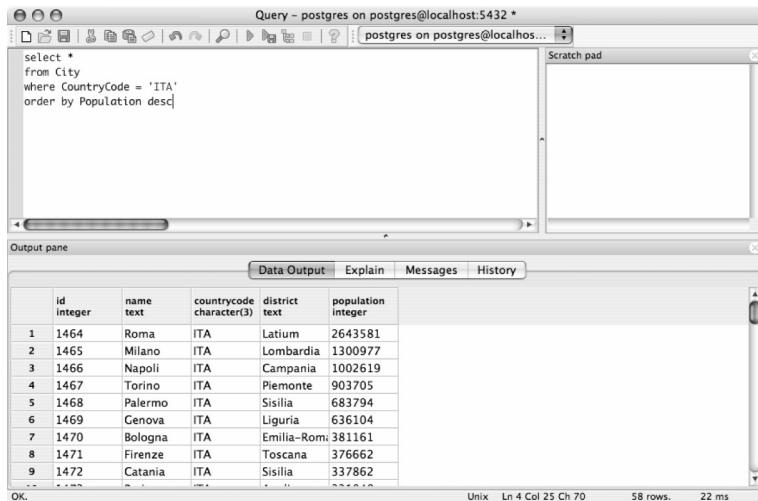


Figura C.2 L'interfaccia di pgAdmin per la definizione di query

La finestra ha un'area di testo editabile come componente principale e inoltre prevede un certo numero di pannelli, una tool bar e un menu. L'uso normale dello strumento prevede di scrivere il testo SQL della query nella componente principale. Il pannello di output (*Output pane*) è responsabile di mostrare il risultato dell'esecuzione del comando. Lo strumento permette anche di analizzare la struttura del piano di esecuzione della query generato internamente dall'ottimizzatore di Postgres, in diversi formati (si analizzano questi aspetti nel Capitolo 11); queste funzioni sono particolarmente importanti per un uso avanzato di Postgres.

Un altro bottone della finestra principale consente invece di visualizzare il contenuto della tabella. Nella Figura C.3 si mostra la finestra con parte del contenuto di una tabella. La finestra permette di specificare nella visualizzazione dei criteri di filtraggio o ordinamento. È inoltre possibile limitare la visualizzazione a un numero limitato di tuple.

### C.3 Costruzione di una base di dati d'esempio

Postgres rappresenta uno dei sistemi relazionali con il maggior grado di compatibilità con lo standard SQL. Oltre al supporto assai esteso per i costrutti definiti nello standard, Postgres presenta alcune forme sintattiche che non rientrano in SQL-3, la cui presenza è giustificata dalla necessità di offrire servizi di gestione dei dati coerenti con l'architettura interna del sistema.

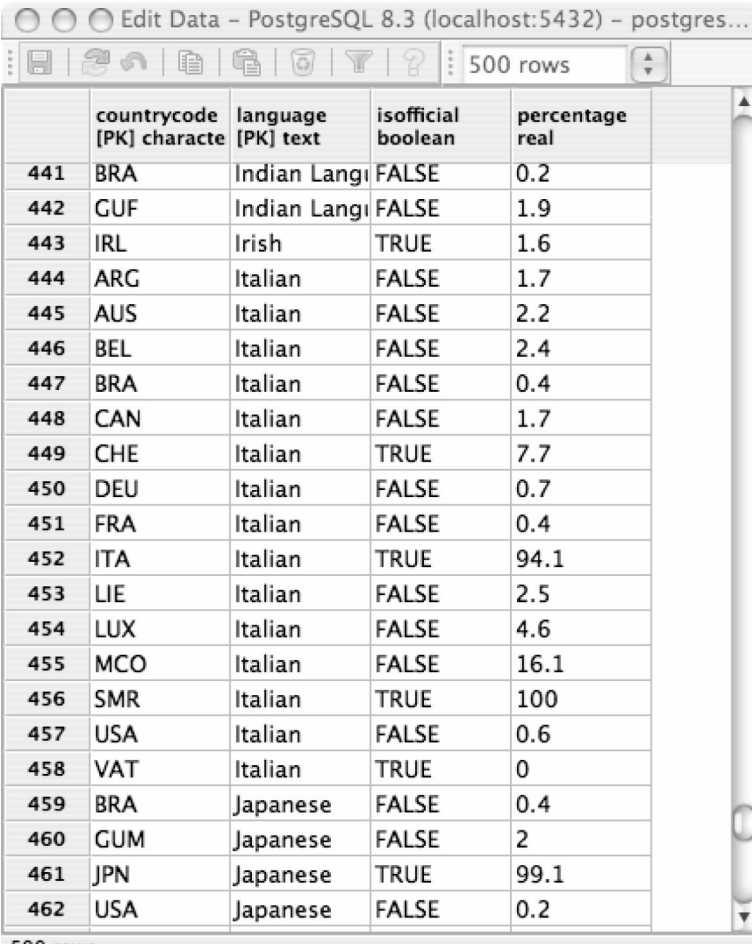
Per quanto riguarda l'insieme di domini di base, Postgres offre una ampia varietà di scelta, che va ben al di là dei classici domini che fanno parte dello standard SQL. Inoltre, Postgres si caratterizza per essere un sistema espandibile, con alcune caratteristiche delle basi di dati a oggetti e il sistema di domini presenta una particolare flessibilità. Utilizzando l'interfaccia a finestre di `pgAdmin` per la costruzione di schemi di tabelle, è possibile osservare un elenco di alcune decine di domini come possibilità di scelta per il dominio di un singolo attributo.

In generale, la costruzione di uno schema di base di dati può fare uso dello strumento `psql`, con l'immissione da terminale o da file della sequenza di comandi SQL in grado di definire lo schema; in alternativa, si può usare l'interfaccia interattiva di `pgAdmin`, che permette di minimizzare l'immissione di testo ed è in grado di ricostruire internamente il codice SQL necessario per la definizione di una certa struttura dati.

In questa appendice, piuttosto che descrivere passo-passo la costruzione di una base di dati d'esempio, assumiamo che sul sistema venga installata una delle basi di dati pubbliche che fanno parte della distribuzione estesa di Postgres. Tra queste basi di dati, scegliamo la base di dati *World*, che consiste di tre tabelle:

```
CITY(Id, Name, CountryCode, District, Population),
COUNTRY(Code, Name, Continent, Region, SurfaceArea,
        IndepYear, Population, LifeExpectancy, Gnp,
        GnpOld, LocalName, GovernmentForm, HeadOfState,
        Capital, Code2),
COUNTRYLANGUAGE (CountryCode, Language, IsOfficial, Percentage)
```

che descrivono dati sintetici di tipo geo-economico. Per esempio, la tabella *CITY* contiene la descrizione di circa 4000 città del mondo, tra cui la descrizione delle prime 58 città italiane per numero di abitanti. La base di dati è ricostruibile invocando l'esecuzione dello script "world.sql", tramite lo strumento `psql` o `pgAdmin`.



	countrycode [PK] characte	language [PK] text	isofficial boolean	percentage real
441	BRA	Indian Langi	FALSE	0.2
442	GUF	Indian Langi	FALSE	1.9
443	IRL	Irish	TRUE	1.6
444	ARG	Italian	FALSE	1.7
445	AUS	Italian	FALSE	2.2
446	BEL	Italian	FALSE	2.4
447	BRA	Italian	FALSE	0.4
448	CAN	Italian	FALSE	1.7
449	CHE	Italian	TRUE	7.7
450	DEU	Italian	FALSE	0.7
451	FRA	Italian	FALSE	0.4
452	ITA	Italian	TRUE	94.1
453	LIE	Italian	FALSE	2.5
454	LUX	Italian	FALSE	4.6
455	MCO	Italian	FALSE	16.1
456	SMR	Italian	TRUE	100
457	USA	Italian	FALSE	0.6
458	VAT	Italian	TRUE	0
459	BRA	Japanese	FALSE	0.4
460	GUM	Japanese	FALSE	2
461	JPN	Japanese	TRUE	99.1
462	USA	Japanese	FALSE	0.2

500 rows.

**Figura C.3** L'interfaccia di pgAdmin per l'accesso al contenuto di una tabella

## C.4 Supporto dello standard SQL

Facendo riferimento alla trattazione del linguaggio SQL del Capitolo 4 e del Capitolo 9, Postgres offre il supporto per grandissima parte dei costrutti. Tra le funzioni non gestite, segnaliamo l'assenza del supporto per la clausola `check` nella definizione delle viste e il vincolo, non presente nello standard SQL, di non usare sottoquery nei vincoli costruiti con la clausola `check`. A parte queste lievi eccezioni, la definizione delle query e delle viste segue quindi la sintassi dello standard SQL. Per esempio, moltiplicando la popolazione di una città per la percentuale di persone di madre lingua italiana nella nazione, si ottiene una stima del numero di persone italiane nella città.



Si può quindi costruire la seguente query che estrae le città al di fuori dell'Italia con persone che parlano italiano, ordinate in base al loro numero:

```
select Name, City.CountryCode, District,
       Population*Percentage/100
from City join CountryLanguage
  on City.CountryCode = CountryLanguage.CountryCode
where Language = 'Italian'
   and City.CountryCode != 'ITA'
order by Population*Percentage desc
```

La seguente query invece estrae per ogni nazione il rapporto tra la popolazione della nazione e la somma degli abitanti delle sue città presenti nella base di dati, presentando i risultati in ordine decrescente del valore del rapporto.

```
select Country.Name, sum(City.Population)*100/Country.Population
from City join Country on City.CountryCode = Country.Code
group by Country.Name, Country.Code, Country.Population
order by sum(City.Population)*100/Country.Population desc
```

Infine, la definizione di una vista che rappresenti le città europee con almeno un milione di abitanti può avvenire con il seguente comando SQL:

```
create view LargeEuropeanCity as
  select City.*
  from City join Country
    on (City.CountryCode = Country.Code)
 where City.Population > 1000000
    and Continent = 'Europe'
```

## C.5 Funzionalità evolute

Presentiamo ora sinteticamente alcune delle caratteristiche evolute di Postgres, che possono essere di interesse per la realizzazione di applicazioni sofisticate. Mostriamo le caratteristiche principali dell'estensione procedurale, l'integrazione con i linguaggi di programmazione di alto livello e la gestione di trigger. Alcuni dei principi che sono alla base della costruzione di questi servizi sono descritti nel secondo volume. Si ritiene comunque utile fornire un inquadramento di queste funzionalità, piuttosto che limitarsi ai soli aspetti trattati in questo primo volume.

### C.5.1 Estensione procedurale

Come descritto nel Paragrafo 5.3, i sistemi relazionali spesso offrono il supporto per un'estensione procedurale di SQL, consentendo la costruzione di procedure che possono essere gestite direttamente dal sistema, venendo incontro alle esigenze degli sviluppatori che debbano realizzare un'applicazione che necessita di svolgere un certo numero di passi di trasformazione relativamente semplici su dati che sono inseriti in ampie collezioni. In questi contesti, si osserva spesso un notevole vantaggio in termini di efficienza del sistema se si riesce a tenere la computazione vicina ai dati stessi, all'interno del motore relazionale.

L'estensione procedurale di SQL riconosciuta da Postgres va sotto il nome di *PL/pgSQL*. Il nome si ispira a PL/SQL, il nome dell'estensione procedurale di Oracle Server.



PL/pgSQL si ispira non solo nel nome a PL/SQL, bensì anche nell'insieme di funzionalità e nella struttura sintattica degli arricchimenti procedurali. L'architettura di Postgres è fortemente modulare e per poter utilizzare il linguaggio PL/pgSQL è necessario averlo installato sul sistema, lanciando dalla shell il comando `createlang plpgsql`

*NomeDatabase.* La necessità di caricare esplicitamente l'estensione procedurale è legata alla scelta di Postgres di mantenere un'elevata flessibilità, consentendo un facile arricchimento del sistema, che può infatti essere esteso in modo relativamente facile con il supporto per altri linguaggi.

Mostriamo alcuni semplici esempi del funzionamento di questa estensione, sulla base di dati di riferimento per questo capitolo. Il primo esempio mostra una procedura che estrae il nome della nazione in cui è parlato dal maggior numero di persone il linguaggio che viene passato come parametro.

```
create function
  NazionePiuParlanti(Linguaggio CountryLanguage.Language%type)
  returns Country.Name%type as
$corpo$
declare Nazione Country.Name%type;
begin
  select C.Name into Nazione
  from Country C join CountryLanguage CL
    on C.Code=CL.CountryCode
  where CL.Language = Linguaggio and
    C.Population * CL.Percentage >= all
      (select Cl.Population * CL1.Percentage
       from Country Cl join CountryLanguage CL1
         on Cl.Code=CL1.CountryCode
        where CL1.Language = Linguaggio);
  return Nazione;
end
$corpo$
language 'plpgsql';
```

Le prime righe della procedura `NazionePiuParlanti()` descrivono i parametri di ingresso e uscita; in questo caso la procedura riceve in ingresso un valore del dominio dell'attributo `Language` della tabella `COUNTRYLANGUAGE` e restituisce un valore dello stesso tipo dell'attributo `Name` della tabella `COUNTRY`. Viene poi il corpo della procedura, racchiuso da una coppia di stringhe identiche racchiuse dal carattere `$`; in questo caso il corpo della procedura è racchiuso tra due coppie della stringa `$corpo$`. Nel corpo della procedura si dichiara inizialmente la variabile `Nazione` e si descrive poi la query SQL che dovrà essere eseguita all'esecuzione della funzione. Nella query compaiono riferimenti al parametro di ingresso `Linguaggio`, che assumerà al momento dell'esecuzione il valore attuale, secondo il classico paradigma di invocazione di procedure comune a tutti i linguaggi di programmazione. L'istruzione `return` restituisce il valore prodotto dalla funzione. Il codice della procedura termina con la specifica del linguaggio utilizzato per la scrittura del corpo della funzione, in questo caso PL/pgSQL. L'invocazione della procedura potrà avvenire nel contesto di una query SQL in ogni ambito nel quale è accettabile un valore del dominio. È una query corretta, per esempio, `select NazionePiuParlanti('Italian')`, così come la seguente, che restituisce per ogni lingua il numero totale di parlanti nelle diverse nazioni, il nome della nazione con il maggior numero di parlanti e la popolazione di questa nazione:

```

select Language, sum(Population*Percentage/100),
       NazionePiuParlanti(Language), C2.Population
from CountryLanguage join Country C1 on CountryCode=C1.Code,
       Country C2
where C2.Name=NazionePiuParlanti(Language)
group by Language, C2.Population
order by sum(Population*Percentage) desc

```

Il seguente frammento di codice introduce una funzione che recupera le città italiane e le restituisce all'ambiente chiamante. All'interno della procedura, l'insieme delle tuple viene considerato una tupla alla volta.

```

create or replace function cittaItaliane()
returns setof city as
$$
declare c city%rowtype;
begin
  for c in select * from city
    where countryCode = 'ITA'
  loop
    if c.Population > 1000000
      return next c.Id | c.Name | c.District | 'Large'
    else
      return next c.Id | c.Name | c.District | 'Small'
    end loop;
  return;
end
$$
language 'plpgsql' ;

```

La procedura fa uso al suo interno di strutture di controllo, con una sintassi molto simile a quella usata in PL/SQL. L'invocazione della procedura può avvenire dovunque può essere valutata un'espressione che restituisce un insieme di tuple, come nell'ambito di una clausola `from`. Per esempio, si può scrivere:

```

select * from cittaItaliane();

```

## C.5.2 Integrazione con altri linguaggi di programmazione

Come detto, Postgres si caratterizza per avere un'architettura esplicitamente aperta e configurabile. Uno degli obiettivi di disegno iniziale era proprio la costruzione di un sistema relazionale flessibile, in grado di interagire in molti modi con l'ambiente informatico esterno e con una forte apertura rispetto alla rigidità che caratterizzava i primi sistemi relazionali. Il disegno di Postgres si è dimostrato coerente con l'evoluzione dei sistemi relazionali i quali di norma offrono oggi questi servizi evoluti. Un aspetto rilevante, dal punto di vista della flessibilità del sistema e della sua capacità di interagire con il resto dei componenti di un sistema informatico, consiste nella capacità di integrare con i normali linguaggi di programmazione.

Postgres offre diverse modalità di integrazione. La modalità più semplice si basa sull'uso di SQL Embedded, che viene realizzato in Postgres rispettando in modo fedele lo standard SQL-2. La descrizione riportata nella Sezione 9.1 è direttamente applicabile in questo contesto. È lo strumento di preprocessing che converte un programma C con all'interno codice SQL embedded in un programma C direttamente compilabile. Postgres offre anche una libreria `libpq` che fornisce un'interfaccia diretta di dialogo

tra un programma C/C++ e l'ambiente Postgres. Il sistema mette inoltre a disposizione librerie e moduli che consentono di interagire con il sistema tramite JDBC e ODBC. Infine, il sistema presenta anche alcuni componenti specifici che consentono di realizzare una stretta integrazione tra i servizi della base di dati e linguaggi quali Python, Perl e Tcl.

Al di là di questa varietà di servizi, che rientrano nelle caratteristiche di molti sistemi relazionali maturi, Postgres si caratterizza per la flessibilità che dimostra nell'integrazione, consentendo non solo di invocare i servizi del motore relazionale in modo sofisticato all'interno di applicazioni scritte in linguaggi di alto livello, ma anche di invocare direttamente dall'interno del sistema relazionale funzioni definite dall'utente utilizzando questi linguaggi di programmazione. Questo servizio offre quindi la possibilità a utenti esperti di estendere in modo relativamente facile il comportamento del sistema relazionale. Non è un caso che Postgres sia la piattaforma più utilizzata nell'ambito della ricerca per esplorare e dimostrare nuove funzionalità dei sistemi relazionali.

L'introduzione di funzioni esterne prevede di inserire nell'esecuzione di accessi ai dati il codice compilato di funzioni C che devono essere state preparate in modo tale da rispettare le convenzioni di gestione dei dati all'interno del motore relazionale. Gli aspetti tecnici da gestire sono relativamente complicati e vanno oltre il livello di dettaglio che caratterizza questa appendice. Al di là dell'intrinseca complessità associata allo sfruttamento di questi servizi, si può osservare che, grazie al suo disegno e alla sua natura di sistema open-source, Postgres presenta un grado di flessibilità non comune e può essere uno strumento interessante per la realizzazione di applicazioni sofisticate.

### C.5.3 Gestione di trigger

Chiudiamo l'analisi delle funzioni evolute di Postgres descrivendo la modalità di gestione dei trigger. I trigger sono stati descritti nella Sezione 5.4 di questo testo. In questa appendice mostriamo sinteticamente le caratteristiche specifiche della gestione dei trigger offerta da Postgres.

La sintassi è la seguente:

```
CREATE TRIGGER Nome
{ BEFORE | AFTER } { Evento [ OR . . . ] }
ON NomeTabella
[ FOR [ EACH ] { ROW | STATEMENT } ]
EXECUTE PROCEDURE NomeFunzione ( Argomenti )
```

Le differenze principali rispetto ai trigger SQL standard sono l'assenza della condizione e la restrizione per cui l'azione del trigger consiste sempre nell'invocazione di una procedura, scritta in qualunque linguaggio, e non nell'esecuzione di istruzioni SQL. Nell'esempio si mostra una procedura di tipo `trigger` scritta in *PL/pgSQL*, ma Postgres ammette anche l'uso di funzioni C, opportunamente costruite. Le procedure C invocate nell'azione dei trigger devono essere progettate gestendo diversi parametri di interfaccia che devono essere tenuti in attenta considerazione, rendendo relativamente complicata la scrittura del codice. Un esempio di trigger Postgres è il seguente.

```

create function ControllaCitta()
    returns trigger as $ControllaCitta$
begin
    -- Controlla che il valore di continente non sia nullo
    if new.Continent is null then
        raise exception 'Continente non puo' essere null';
    end if;
    -- Controlla se e' un valore accettabile
    if new.Population <= 0 then
        raise exception '% deve avere una popolazione positi-
va',
            new.Name;
    end if;
end;
$ControllaCitta$ language plpgsql;
create trigger ControllaCitta
    before insert or update on City
    for each row
    execute procedure ControllaCitta();

```

I trigger possono essere sia *row-level*, con un'invocazione del trigger per ogni tupla distinta, sia *statement-level*, con un'invocazione unica del trigger in risposta al comando che ha generato l'evento, indipendentemente dal numero di tuple modificate dal comando. Il sistema permette anche di specificare sia trigger *before*, sia *after*, senza porre restrizioni nella sintassi all'azione dei trigger before (il manuale del sistema consiglia giustamente prudenza). Per i trigger before row-level, Postgres assume che la procedura invocata nell'azione di un trigger insert o update restituisca una tupla, che sostituirà la tupla originariamente prodotta dal comando che ha attivato il trigger. Per quanto riguarda l'ordine di esecuzione dei trigger, qualora ve ne siano diversi pronti, Postgres segue l'ordine alfabetico sul nome del trigger. Postgres non prevede alcun limite esplicito al numero di invocazioni in cascata dei trigger; è quindi piena responsabilità del programmatore garantire la terminazione dei trigger.

Rispetto alla definizione dei trigger prevista dallo standard SQL-3, in Postgres è possibile fare riferimento alle variabili `old` e `new` nell'ambito dei trigger row-level, mentre non è ancora previsto il supporto per `old table` e `new table` nei trigger statement-level. Le variabili `old` e `new` inoltre non possono essere ridenominate. Non è poi possibile specificare trigger che reagiscono a eventi di update su specifici attributi. In SQL-3 l'azione può presentare più componenti da eseguire in sequenza, senza la necessità di introdurre un'apposita funzione. In generale, i limiti che presentano i trigger in Postgres sono di entità relativamente lieve e non pongono particolari ostacoli nello sviluppo di applicazioni. A causa dell'assenza di un meccanismo che limiti il numero di esecuzioni dei trigger, bisogna però prestare molta attenzione nella scrittura del codice. Alcune delle limitazioni dei trigger sono giustificate dalla presenza in Postgres di un meccanismo alternativo, le *rules*, che rappresentano uno strumento per la riscrittura di comandi SQL, che sono alla base di diversi meccanismi evoluti. Rimandiamo il lettore interessato alla consultazione della documentazione del sistema.

