Capitolo 6

Le macro e i moduli

Nei primi cinque capitoli abbiamo passato in rassegna gli oggetti Access che costituiscono un database: le tabelle e le query, e quelli che si utilizzano per accedere al database, quindi le maschere e i report. Combinando opportunamente un certo numero di questi oggetti si crea un'applicazione gestionale fondata su un database, il prodotto finito da ottenere con Access.

Affinché un'applicazione di questo genere dia risultati professionali è necessario, però, che i vari oggetti fin qui descritti siano dotati di funzionalità specifiche, che si costruiscono scrivendo programmi con i due linguaggi di programmazione interni ad Access, il Visual Basic for Applications (in sigla VBA) e lo Structured Query Language, noto anche come SQL. Questi programmi, detti in gergo *routine*, risiedono nelle macro e nei moduli. Prima di descrivere le caratteristiche di questi due oggetti vediamo per quale ragione servono altre funzionalità in aggiunta a quelle, pur numerosissime, che caratterizzano gli oggetti Access esaminati fin qui.

Le funzionalità aggiuntive

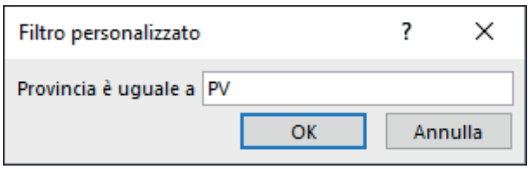
Apriamo in visualizzazione *Maschera* una delle maschere che abbiamo creato per prova nel Capitolo 4, con la quale possiamo accedere alla tabella TBLINDIRIZZI; questa tabella dimostrativa contiene una sessantina di indirizzi, che in un'applicazione reale potrebbero essere molti di più, centinaia o migliaia.

Supponiamo di aver bisogno di rintracciare una persona della quale ricordiamo soltanto che abita nella provincia di Pavia. Scorrere con la maschera tutti gli indirizzi a uno a uno, cercando quelli che hanno la sigla *PV* nella casella PROVINCIA può essere un'impresa lunga e noiosa. Se

potessimo ridurre l'ambito della ricerca e visualizzare soltanto i record della provincia di Pavia, l'operazione durerebbe sicuramente meno.

A questo scopo, eseguiamo le seguenti operazioni:

- con un clic destro sul controllo che contiene la sigla della provincia facciamo uscire un menu contestuale, che scorriamo fino alla voce *Filtri testo*;
- un clic su *Filtri testo* apre un sottomenu con una serie di comandi di confronto fra i quali scegliere;
- selezioniamo la voce *Uguale a* e otteniamo l'apertura di una minuscola finestra di comando come quella che vediamo qui di seguito, nella quale digitiamo **PV**, la sigla della provincia che ci interessa;



- un clic su *OK* chiude la finestra di dialogo e attiva il filtro.

Adesso la nostra maschera si presenta come nella Figura 6.1.

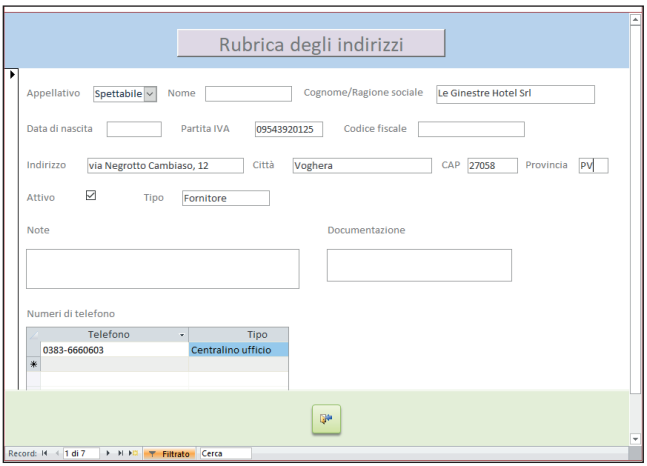


Figura 6.1 La maschera dove è stato impostato un filtro mostra soltanto 7 record invece di 62.

Alla destra dei pulsanti di scorrimento compare ora una dicitura col numero dei record che hanno il valore *PV* nella casella PROVINCIA, seguito dall'indicazione (*Filtrato*) per segnalare che i record sottostanti alla maschera sono più numerosi di quelli accessibili in questo momento.

Con i record potenziali ridotti a sette facciamo molto in fretta a trovare quello che ci interessa, scorrendo soltanto quelli filtrati. Per disattivare il filtro e tornare ad avere accesso a tutti i record della tabella TBLINDIRIZZI ci basta fare clic sulla dicitura *Filtrato* che sta appena a destra dei pulsanti di scorrimento.

Fin qui tutto bene: conosciamo Access e sappiamo come accedere ai comandi che si trovano nelle varie schede della barra multifunzione. Ma per usare un'applicazione creata con Access non dovrebbe essere necessario conoscere Access: all'utente finale, cioè all'operatore della società cliente che ci ha commissionato l'applicazione, interessa poter consultare i dati con maschere chiare e facili da usare e se gli facesse comodo filtrare i record in base a un determinato criterio non dovrebbe andare a cercare comandi Access da qualche parte, ma dovrebbe averli a disposizione, in forma chiara e intuitiva, nella maschera stessa con la quale sta lavorando. È in questo spirito, d'altronde, che abbiamo inserito un pulsante di comando nel piè di pagina di questa maschera (si veda il paragrafo "Pulsante di comando" del Capitolo 4), in modo che l'operatore possa chiuderla agevolmente, senza cercare il comando Access di chiusura nella barra del titolo, in un menu contestuale o nella barra multifunzione.

Attivare e disattivare un filtro in base al valore di un campo predefinito, chiudere una maschera con un pulsante di comando personalizzato e magari aprirne contestualmente un'altra o avviare la stampa di un report selezionandolo entro un gruppo di opzioni collocato in una maschera, sono questi i tipi di funzionalità aggiuntive che fanno di un file Access un'applicazione gestionale e per crearle si deve ricorrere alle macro o ai moduli.

Vediamo in primo luogo come si creano le macro e che cosa possono dare, successivamente esamineremo i moduli.

Creare una macro

Le macro sono oggetti Access composti da *azioni*, scelte in un elenco prestabilito chiamato *Catalogo azioni* e inserite in una finestra di composizione. Per "azione" si intende un'operazione che può essere

molto semplice (aprire o chiudere una maschera) o molto complessa (esportare una o più tabelle dal database corrente in un altro database o in un altro computer collegato in rete). Una macro può articolarsi in più azioni, che di norma vengono eseguite automaticamente tutte di seguito, nell'ordine in cui sono state collocate nella finestra di composizione. Ricordiamo anche, per completare la nomenclatura delle macro, che un certo numero di azioni complesse è articolato in passi; inoltre, con l'eccezione di pochi casi di azioni estremamente semplici, le macro utilizzano valori e parametri – alcuni obbligatori, altri facoltativi – sui quali basare le proprie azioni.

Si accede alle macro dalla barra multifunzione, eseguendo il comando *Macro* nel gruppo *Macro e codice* della scheda *Crea*.

Nella finestra *Database* si apre una finestra di composizione vuota, che porta il titolo predefinito Macro1, sulla destra compare un pannello intitolato *Catalogo azioni*, che elenca le azioni disponibili aggregate in gruppi e presentate graficamente con una variante della simbologia usata in Windows 10 per visualizzare cartelle e sottocartelle nella finestra di Esplora file: ogni gruppo è associato all'icona di una cartella, affiancata a sinistra da un pulsante con un triangolo bianco che punta verso destra; un clic su quel triangolo fa sgranare verso il basso l'elenco delle azioni di quel particolare gruppo, il triangolo bianco diventa un triangolo nero che punta verso il basso: se gli elementi che vengono visualizzati subito sotto sono sottogruppi hanno anch'essi un triangolo bianco a sinistra che se premuto fa uscire la lista degli elementi di quel sottogruppo.

Un clic su un triangolino nero fa rientrare l'elemento nel gruppo al quale appartiene e ripristina la visualizzazione del triangolino bianco. Tutto questo si fa prima a provarlo con qualche clic (tanto non succede ancora niente) che a descriverlo.

Contestualmente all'apertura della macro, nella barra multifunzione si rende disponibile una nuova scheda, chiamata *Strumenti macro/Progettazione*. La scheda contiene tre gruppi di comandi: *Strumenti, Comprimi/Espandi e Mostra/Nascondi*.

Nel primo gruppo troviamo tre comandi:

- *Esegui*, identificato dall'icona del punto esclamativo rosso, che esegue la macro contenuta nella finestra di composizione, se la macro è già stata salvata con un nome;
- *Passo a passo* esegue le azioni della macro un passo per volta per consentire di individuare eventuali errori durante l'esecuzione;

- *Converti macro in Visual Basic* fa esattamente quello che dice: crea un modulo e vi inserisce una routine VBA che corrisponde esattamente alle funzionalità definite dalle azioni della macro convertita.

Il gruppo *Comprimi/Espandi* contiene quattro comandi per disporre in modo compresso o espanso l'elenco delle azioni contenute nella finestra della macro.

Il gruppo *Mostra/Nascondi* contiene due comandi, *Catalogo azioni* e *Mostra tutte le azioni*. Il primo viene eseguito automaticamente quando si avvia la creazione di una macro e fa aprire la finestra di dialogo *Catalogo azioni* nella finestra *Database*; se si esegue questo comando quando il *Catalogo azioni* è aperto si chiude la finestra corrispondente.

Access considera certe azioni macro potenzialmente pericolose per la sicurezza o l'integrità degli oggetti database, e quindi tali azioni non vengono elencate nel catalogo: il comando *Mostra tutte le azioni* fa sì che nel *Catalogo azioni* vengano elencate anche le azioni considerate rischiose, il cui nome compare affiancato dall'icona convenzionale del pericolo (un triangolo con un punto esclamativo nel centro), mentre i nomi delle azioni non rischiose sono affiancati dall'icona di una saetta. Dal momento che diamo per scontato che tutti coloro che usano Access per creare applicazioni gestionali siano maggiorenni e vaccinati, suggeriamo di eseguire sempre il comando *Mostra tutte le azioni* in modo da avere nel catalogo l'elenco completo delle azioni macro.

Nella Figura 6.2 vediamo la schermata completa di tutti gli elementi per creare una macro.

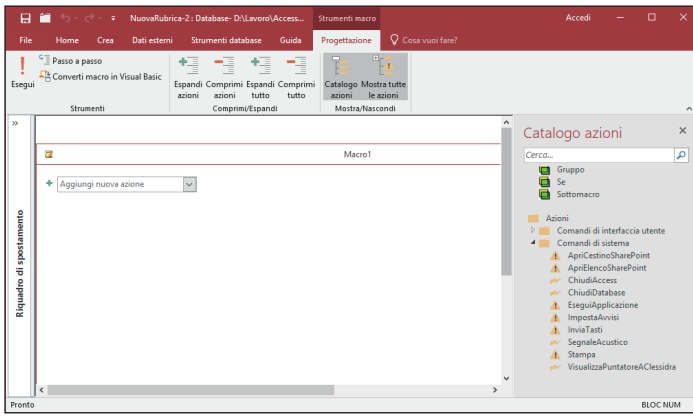


Figura 6.2 La finestra Database quando si prepara una macro.

Diamo adesso un po' di concretezza a tutto quanto precede costruendo la macro mcrSaluto che abbiamo mostrato nella Figura 1.9 del Capitolo 1.

Nel *Catalogo azioni* facciamo clic sul triangolo bianco accanto al gruppo *Comandi di interfaccia utente*, ottenendo così l'elenco delle azioni, che in quel gruppo appaiono tutte caratterizzate come "sicure" (icona della saetta accanto al loro nome). Un clic sull'azione *FinestraMessaggio* fa uscire al piede della finestra *Catalogo azioni* una breve ma chiara descrizione dell'effetto di questa azione.

Un doppio clic sul nome dell'azione apre, nella finestra di composizione della macro, una griglia formata da caselle di testo e caselle combinate, nelle quali si digitano o si scelgono i dati e i parametri necessari per eseguire questa particolare azione.

Vogliamo fare in modo che l'esecuzione della macro visualizzi una finestra di messaggio, quindi digitiamo nella casella di testo *Messaggio* il testo del nostro messaggio: questo dato è, ovviamente, indispensabile se intendiamo ottenere un risultato plausibile. Siamo liberi, volendo, di scrivere un breve testo che faccia da titolo per la finestra di messaggio, digitandolo nella casella di testo *Titolo* e possiamo selezionare nelle due caselle combinate *SegnaleAcustico* e *Tipo* uno dei valori predefiniti, che nel primo caso può essere un *Sì* o un *No* e nel caso del *Tipo* possiamo scegliere fra le icone da visualizzare nella finestra della macro: *Nessuna*, *Messaggio critico*, *Avviso !, Avviso ?* e *Informazione*.

Fatte le scelte opportune, dobbiamo salvare la nuova macro per poterla eseguire; questo si fa con un clic sul pulsante *Salva*, quello col simbolo del dischetto nella barra di accesso rapido. Ci viene chiesto di scrivere un nome, digitiamo **mcrSaluto** e infine premiamo il pulsante con il punto esclamativo rosso che sta in *Strumenti macro/Progettazione/Strumenti*. Nella Figura 6.3 vediamo il risultato di questo lavoro di preparazione e la finestra di messaggio prodotta dalla esecuzione della macro.

Abbiamo eseguito direttamente la macro mcrSaluto tanto per provarla, ma di norma le macro vengono eseguite a partire da una maschera o da un report quando si verifica un particolare evento. Di che cosa si tratta? Nel gergo di Access sono chiamati *eventi* un vasto numero di azioni e accadimenti che si verificano automaticamente o per scelta dell'utente-operatore. Sono eventi l'apertura e la chiusura di una maschera o di un report, la modifica del contenuto di una casella di testo, la selezione di una voce in una casella combinata o il clic su un pulsante di comando.

Le maschere, i report e i controlli che vi sono inseriti sono tutti predisposti per tener conto di un certo numero di eventi o azioni che accadono: per ciascuno di questi oggetti si trova l'elenco degli eventi specifici che lo caratterizzano nella scheda *Evento* della sua *Finestra delle Proprietà*, come possiamo vedere nella Figura 6.4.

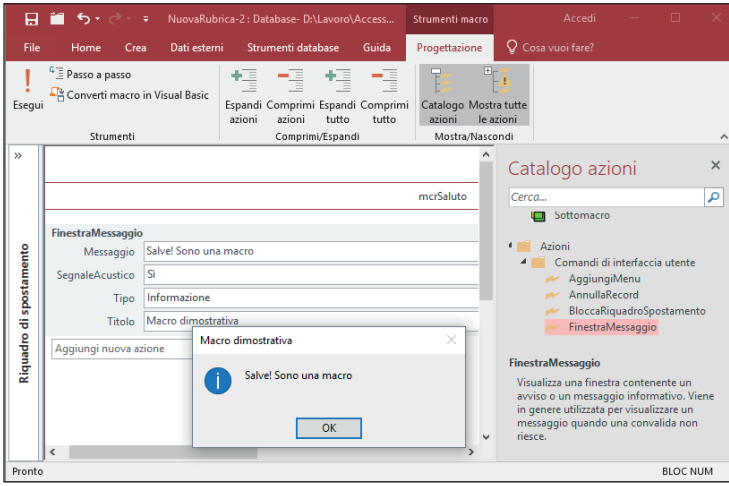


Figura 6.3 La finestra di composizione di una nuova macro e il suo risultato.

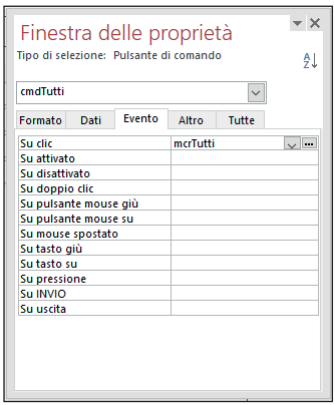


Figura 6.4 La scheda Evento di un pulsante di comando elenca gli eventi che quel controllo è in grado di riconoscere.

Nella figura si vede che per l'evento *Su clic* sul pulsante di comando chiamato CMDTUTTI è stato impostato il richiamo a una macro chiamata MCRTUTTI: con questa impostazione, quando si fa clic su quel pulsante mentre la maschera in cui si trova è aperta in visualizzazione *Maschera*, viene eseguita la macro MCRTUTTI.

Lavorare con le macro

Sviluppiamo ora un semplice esempio per capire come rendere più agevole l'uso di una maschera con l'aiuto di una macro.

Fra le molte azioni macro ve n'è una che si chiama *ApplicaFiltro* e il suo nome lascia capire che consente di ottenere lo stesso risultato al quale si arriva dando il comando *Home/Ordina e filtra/Filtro* dalla barra multifunzione. L'azione prevede tre parametri o argomenti, come si chiamano in gergo: *Nome filtro*, *Nome controllo* e *Condizione WHERE*: i primi due sono facoltativi, mentre il terzo è obbligatorio. Il nome di questo argomento allude a una clausola dell'istruzione SELECT dello Structured Query Language, che si articola in questo modo:

```
SELECT [elenco di campi da selezionare]

FROM [elenco delle origini dei campi da selezionare]

WHERE [condizioni da rispettare per eseguire la selezione]
```

Quindi, con l'argomento chiamato *Condizione WHERE* si specifica il modo in cui eseguire questa azione macro. Abbiamo visto prima (all'inizio del paragrafo "Le funzionalità aggiuntive") come impostare un filtro personalizzato eseguendo una serie di operazioni col mouse: nell'azione *ApplicaFiltro*, l'argomento *Condizione WHERE* serve per ottenere lo stesso risultato.

Serviamoci dello stesso esempio utilizzato per parlare dei filtri e vediamo come si può usare l'azione macro *ApplicaFiltro* per ottenere il risultato della Figura 6.1. In pratica, dovremo costruire una macro basata su questa azione, specificando una *Condizione WHERE* che indica di filtrare i record visualizzati nella maschera FRMINDIRIZZI1 in base alla sigla di una provincia.

Facciamo una copia della maschera FRMINDIRIZZI1, salvandola col nome **frmIndirizziConFiltro**. Per consentire all'utente di selezionare una provincia in base alla quale filtrare i record presentati in questa maschera avremo bisogno di un elenco delle sigle delle province presenti

in tutti i record della tabella TBLINDIRIZZI; in questo elenco i nomi delle diverse province dovranno comparire una sola volta. Possiamo ottenerlo con questa semplice istruzione SQL:

```
SELECT DISTINCT tblIndirizzi.Prov

FROM tblIndirizzi

ORDER BY tblIndirizzi.Prov;
```

La clausola `DISTINCT` fa sì che vengano selezionati solo i nomi univoci. La griglia di struttura della query corrispondente è nella Figura 6.5.

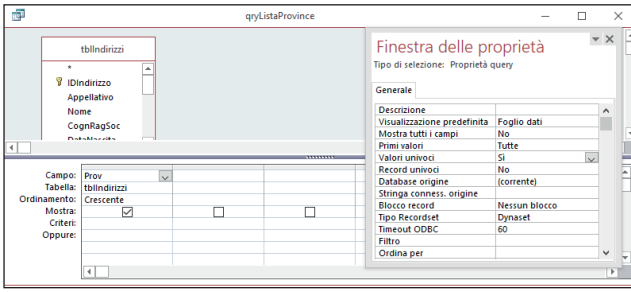


Figura 6.5 La query estrae tutte le sigle delle province, ignorando i valori ripetuti.

Salviamo la query col nome **qryListaProvince** ed eseguiamola per prova: i record della tabella TBLINDIRIZZI sono 62, ma le righe estratte dalla query sono solo 24. Se in futuro, aggiungendo record alla tabella TBLINDIRIZZI, si avranno nel campo PROV valori diversi da quelli già esistenti, la query produrrà un elenco aggiornato con i nuovi valori univoci.

Apriamo in visualizzazione *Struttura* la maschera FRMINDIRIZZICONFILTRO e inseriamo in una zona libera del suo *Corpo* una casella combinata e un pulsante di comando. Impostiamo alcune proprietà dei nuovi controlli:

Proprietà	Casella combinata	Pulsante di comando
Nome elemento	cboSceltaProv	cmdTutti
Etichetta	Scegli Provincia	Mostra tutti
Tipo origine riga	Tabella/query	
Origine riga	qryListaProvince	
Colonna associata	1	
Solo in elenco	Sì	

Salviamo le modifiche e apriamo la maschera così modificata: facendo clic sulla casella combinata cboSceltaProv scenderà l'elenco delle province derivato dall'associazione fra questo controllo e la query che abbiamo creato prima. Un clic sul pulsante di comando non ha, per ora, alcun effetto.

Chiudiamo la maschera e diamo il comando *Crea/Macro e codice/Macro*, ottenendo l'apertura della finestra di composizione di una nuova macro. Nella finestra *Catalogo azioni* individuiamo l'azione *ApplicaFiltro* nella cartella *Filtro/Query/Ricerca* e facciamo doppio clic sul suo nome.

Nella finestra di composizione della nuova macro si inseriscono tre caselle di testo vuote intitolate *Nome filtro*, *Condizione WHERE* e *Nome controllo*. Ignoriamo la prima e la terza perché gli argomenti corrispondenti sono facoltativi e in *Condizione WHERE* digitiamo questa riga di codice:

```
[Prov]=[Maschere]![frmIndirizziConFiltro]![cboSceltaProv]
```

La riga di codice per l'argomento *Condizione WHERE* stabilisce che il filtro va applicato sul campo Prov e il criterio per filtrare è il valore contenuto nel controllo cboSceltaProv della maschera frmIndirizziConFiltro.

Salviamo la macro col nome **mcrScegliProvincia** e chiudiamola senza provarla. Creiamo una seconda macro, selezionando nella cartella *Filtro/Query/Ricerca* del *Catalogo azioni* l'azione *MostraOgniRecord*, che non richiede argomenti, e subito dopo inseriamo l'azione *ImpostaValore*, che si trova nella cartella *Oggetti di database* del *Catalogo azioni*. Questa azione richiede due argomenti: nel primo, chiamato *Elemento*, digitiamo questa stringa di caratteri:

```
[Maschere]![frmIndirizziConFiltro]![cboSceltaProv]
```

e nella casella del secondo argomento, chiamato *Espressione*, digitiamo una coppia di virgolette doppie.

L'azione *MostraOgniRecord* non ha bisogno di argomenti e ha lo stesso effetto del comando *Rimuovi filtro* dato in *Home/Ordina e filtra*.

L'azione *ImpostaValore* assegna un valore a un controllo della maschera, indicato nell'argomento *Elemento* (in questo caso la casella combinata cboSceltaProv della maschera frmIndirizziConFiltro); il valore da impostare è quello digitato per l'argomento *Espressione*, cioè due virgolette doppie senza niente in mezzo, che è una stringa vuota: l'esecuzione delle due azioni di questa macro toglie un filtro, se è impostato, e inserisce una stringa vuota nella casella combinata cboSceltaProv.

Salviamo questa seconda macro col nome MCRTUTTI e chiudiamola. Torniamo alla maschera FRMINDIRIZZICONFILTRO che apriamo in visualizzazione *Struttura* e, dopo aver aperto la *Finestra delle proprietà*, completiamo nel modo seguente le proprietà dei controlli CBOSCELTAPROV e CMDTUTTI:

Controllo	Proprietà	Valore
Casella combinata CBOSCELTAPROV	*Dopo aggiornamento*	MCRSCEGLIPROVINCIA
Pulsante di comando CMDTUTTI	*Su clic*	MCRTUTTI

Le proprietà selezionate appartengono alla categoria delle proprietà *Evento*, e i valori impostati fanno sì che:

- quando si seleziona un valore nella casella combinata, aggiornandone in questo modo il contenuto, venga eseguita la macro MCRSCEGLIPROVINCIA;
- quando si fa clic sul pulsante di comando venga eseguita la macro MCRTUTTI.

Salviamo le modifiche e apriamo la maschera, che si presenterà come nella Figura 6.6: selezionando un valore dalla casella combinata con l'etichetta *Scegli Provincia* si applica automaticamente il filtro corrispondente alla provincia selezionata, che resta attivo fino a quando si preme il pulsante di comando con l'etichetta *Mostra tutti*, che svuota il contenuto della casella combinata e disattiva il filtro, per cui la maschera mostra tutti i record.

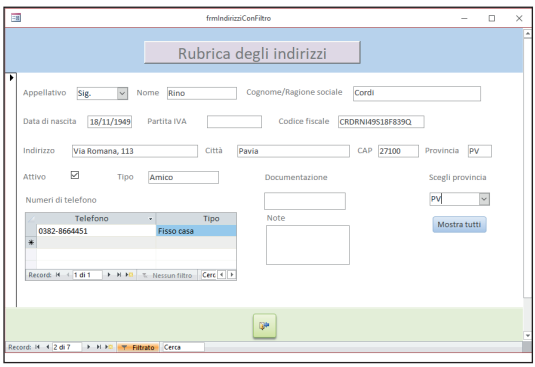


Figura 6.6 La maschera frmIndirizziConFiltro in azione.

Poter ridurre l'ambito di ricerca di un record ai soli indirizzi di una determinata provincia, selezionando la provincia con un semplice clic è già un passo avanti, ma si può fare molto di più.

Nella scheda *Home* della barra multifunzione si può attivare un comando chiamato *Trova* premendo il pulsante con l'icona di una lente di ingrandimento.

Diamo questo comando dopo aver selezionato il campo con il cognome o la ragione sociale. Sulla maschera si sovrappone la finestra di dialogo *Trova e sostituisci*, articolata in due schede, *Trova e Sostituisci*. Selezionando opportunamente le opzioni disponibili nella prima scheda possiamo trovare un record che contiene un determinato valore nel campo selezionato.

La scheda *Trova* presenta quattro caselle combinate e due caselle di controllo, per impostare le opzioni disponibili.

- *Trova* In questa casella combinata si digita il valore da cercare; se sono già state fatte altre ricerche per valori diversi, questi si possono reperire selezionandoli dall'elenco a discesa.
- *Cerca in* Si può selezionare il campo entro il quale cercare il valore (che corrisponde al campo selezionato nel momento in cui si è dato il comando *Trova*) oppure selezionare il nome della tabella, indicando così che la ricerca va fatta in tutti i campi di ogni record.
- *Confronta* Sono disponibili tre opzioni: *Parte del campo*, per un valore che può trovarsi in un punto qualunque del campo; *Campo intero*, per un valore che deve coincidere con l'intero contenuto del campo; *Inizio campo*, per un valore che deve trovarsi all'inizio del campo.
- *Cerca in* Anche qui si hanno tre opzioni: *Tutto*, la ricerca viene eseguita sull'intera tabella; *Giù*, si esplorano soltanto i record successivi a quello selezionato; *Su*, si esplorano soltanto i record che precedono quello selezionato
- *Maiuscole/minuscole* Se questa casella di controllo è selezionata, la ricerca tiene conto delle differenze tra maiuscole e minuscole fra il valore digitato nella casella *Trova* e quelli reperiti nell'area indicata nella casella *Confronta*.
- *Cerca in campi come formattati* Selezionando questa casella di controllo si precisa che il valore da trovare deve avere lo stesso formato del valore indicato in *Trova*.

Se si selezionano le opzioni della scheda *Trova* nel modo che vediamo nella Figura 6.7, un clic sul pulsante *Trova successivo* reperisce un primo record col contenuto che appare evidenziato nella maschera che sta in secondo piano.

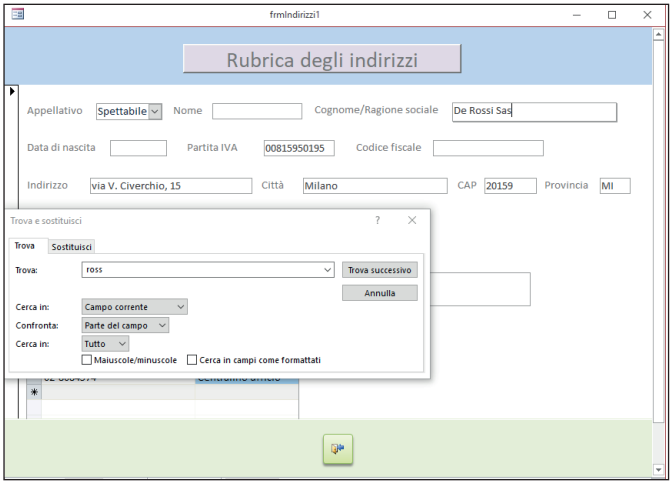


Figura 6.7 Il funzionamento del comando Trova.

Le impostazioni date per la ricerca, nell'esempio illustrato dalla figura, dicono: "Trova un record che contenga in un punto qualunque del campo COGNOME/RAGIONE SOCIALE la stringa *ross*, ignorando la differenza fra maiuscole e minuscole e il formato ed esplorando tutti i record iniziando dal primo"

Come si può vedere, la ricerca ha individuato un record che contiene nel campo COGNOME/RAGIONE SOCIALE la stringa *De Rossi sas*, che nei caratteri che vanno dal quarto al settimo corrisponde al valore da cercare (*ross*).

Ulteriori clic sul pulsante di comando *Trova successivo* fanno riprendere la ricerca fino a quando non si trovano più record che soddisfano le condizioni poste nella scheda *Trova*.

È indiscutibile che *Trova* sia un comando molto potente e flessibile, ma richiede, ancor più del comando *Filtro*, che abbiamo visto prima, una notevole dimestichezza con Access, una dimestichezza che non possiamo dare per scontata in chi userà la nostra applicazione.

Con l'aiuto di un paio di macro e l'aggiunta di una casella di testo e di un pulsante di comando possiamo trasferire direttamente in una maschera tutte le funzionalità del comando *Trova*, rendendole facilmente accessibili a chi usa la maschera, senza dover ricorrere alla barra multifunzione di Access.

Facciamo una copia della maschera FRMINDIRIZZICONFILTRO, salvandola col nome **frmIndirizziConRicerca**, quindi inseriamo nel piè di pagina della nuova maschera una casella di testo e due pulsanti di comando, impostando queste loro proprietà:

	Casella di testo	Etichetta della casella di testo	Pulsante di comando 1	Pulsante di comando 2
Nome elemento	**txtTrova**		**cmdTrova**	**cmdAncora**
Etichetta		**Cognome da trovare:**	**Trova**	**Ancora**
Testo descrizione controllo			**Clic per avviare la ricerca**	**Clic per ripetere la stessa ricerca**

Disponiamo i nuovi controlli nel modo indicato nella Figura 6.8. Approfittiamo dell'occasione per spostare anche il pulsante di comando CMDTUTTI (si veda sopra, la Figura 6.6) dal corpo della maschera nello stesso piè di pagina in cui collochiamo i nuovi controlli.

Figura 6.8 I nuovi controlli inseriti nella sezione Piè di pagina della maschera frmIndirizziConRicerca.

I nomi usati per la casella di testo e per i pulsanti di comando, insieme con le descrizioni che compaiono quando si appoggia il puntatore del mouse sui pulsanti di comando, rendono chiaro all'operatore che per cercare un record con un determinato cognome deve scrivere il cognome da trovare nella casella di testo e premere il pulsante *Trova* per trovare il primo e poi il pulsante *Ancora* per trovarne altri, se ce ne sono.

Perché questo accada dobbiamo creare due macro che vengano eseguite quando si fa clic su uno o l'altro dei due pulsanti di comando.

Per entrambe le macro abbiamo bisogno di due azioni, che si chiamano *VaiAControllo* e *TrovaRecord*. L'azione *VaiAControllo* richiede un solo argomento, il nome del controllo (ovviamente), che per entrambe le macro sarà il nome della casella di testo che contiene il cognome o la ragione sociale; questo controllo si chiama CognRagSoc e lo andremo a scrivere nella casella di testo *Nome controllo* dell'azione *VaiAControllo*. Salviamo la prima macro col nome **mcrPrimo** e la seconda col nome **mcrSuccessivo**.

La seconda azione, *TrovaRecord*, è anch'essa comune a entrambe le nuove macro, richiede ben sette argomenti, che andranno impostati nel modo seguente:

Per la macro mcrPrimo:

Trova	`=[Maschere]![frmIndirizziConRicerca]![txtTrova]`
Confronta	**Parte del campo**
Maiuscole/ minuscole	**No**
Cerca in	**Tutto**
Come formattato	**No**
Solo campo corrente	**Sì**
Trova primo	**Sì**

Per la macro mcrSuccessivo:

Trova	`=[Maschere]![frmIndirizziConRicerca]![txtTrova]`
Confronta	**Parte del campo**
Maiuscole/ minuscole	**No**
Cerca in	**Giù**
Come formattato	**No**
Solo campo corrente	**Sì**
Trova primo	**No**

Alla fine del lavoro di preparazione la finestra della macro mcrPrimo dovrebbe presentarsi come nella Figura 6.9.

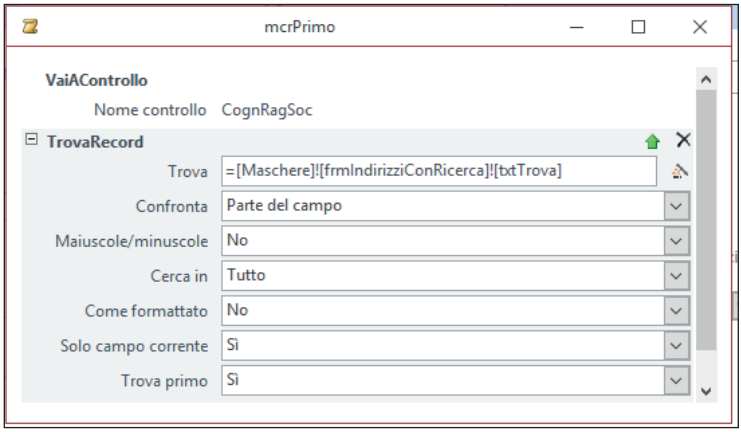


Figura 6.9 La macro mcrPrimo nella finestra di composizione.

Le due macro corrispondono all'esecuzione del comando *Trova* con le impostazioni dell'esempio che abbiamo visto sopra, nel caso della macro mcrPrimo (si veda la Figura 6.7), mentre la macro che abbiamo chiamato mcrSuccessivo esegue lo stesso comando, ma esplorando i record successivi a quello trovato quando è stata eseguita la prima macro.

Si tratta adesso di associare l'esecuzione dell'una o dell'altra macro ai due pulsanti di comando, operazione che si esegue semplicemente impostando le loro proprietà *Su clic* in questo modo:

Pulsante di comando	**Proprietà Evento** *Su clic*
`cmdTrova`	*mcrPrima*
`cmdAncora`	*mcrSuccessivo*

Ancora un piccolo dettaglio: abbiamo spostato il pulsante di comando cmdTutti (quello che ha l'etichetta *Mostra tutti*) insieme con i nuovi controlli nella sezione *Piè di pagina* della maschera: in quella posizione può far comodo premerlo anche quando si vuole annullare l'effetto

del comando di ricerca, oltre che per annullare il filtro basato sul campo della Provincia. Ci basterà aggiungere questa terza azione alle due che avevamo già impostato per la macro mcrTutti:

```
ImpostaValore

Elemento = [Maschere]![frmIndirizziConRicerca]![txtTrova]

Espressione = " "
```

Dopo aver salvato tutte le modifiche, apriamo la maschera frmIndirizziConRicerca in visualizzazione *Maschera*, scriviamo un gruppo di caratteri nella casella di testo con l'etichetta *Cognome da trovare*, premiamo il pulsante di comando con la dicitura *Trova* e otteniamo il risultato che vediamo nella Figura 6.10.

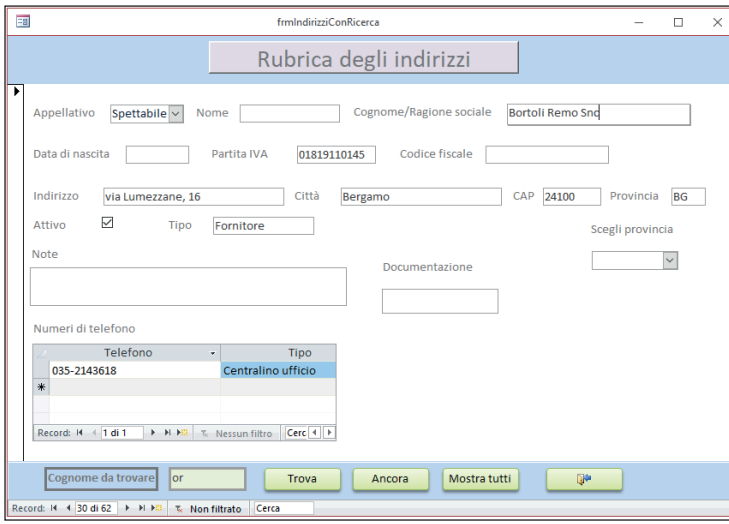


Figura 6.10 Nella maschera arricchita di funzionalità si può cercare un record che contenga un determinato valore nel campo del cognome.

Dai due semplici esempi che abbiamo presentato fin qui si dovrebbe cominciare a capire l'importanza delle funzionalità aggiuntive di questo tipo per migliorare l'interazione fra l'operatore e l'applicazione Access e il ruolo che le macro possono svolgere per creare e attivare questo genere di funzionalità.

Le azioni macro disponibili in Access 2019 sono ottantasei: organizzandole opportunamente in macro si possono ottenere funzionalità di notevole interesse da aggiungere a maschere associate e non associate per renderle più efficaci e più facili da utilizzare.

Pur essendo state progressivamente migliorate a ogni nuova versione, le Guide dei prodotti Office e di Access in particolare continuano a contenere testi farraginosi e spesso difficili da reperire, quindi per studiare le caratteristiche delle azioni macro e farsi un'idea di quello che si può ottenere con questi strumenti è opportuno leggere qualche testo che tratti l'argomento con maggiore chiarezza: a questo scopo, può essere utile il libro Costruire applicazioni con Access*, di Mike Davis, pubblicato da Apogeo, che dedica parecchie decine di pagine alle azioni macro e le descrive una per una in modo semplice e accurato.*

Tutto bene, quindi? Le macro offrono tutti gli strumenti di cui potremmo aver bisogno per sviluppare con Access applicazioni gestionali di qualità professionale? Beh, non esattamente, come vediamo nel prossimo paragrafo.

I limiti delle macro

Torniamo ad aprire la maschera FRMINDIRIZZICONRICERCA che abbiamo appena arricchito con le macro MCRPRIMO e MCRSUCCESSIVO e premiamo il pulsante *Trova* senza aver prima digitato una stringa di caratteri nella casella di testo *Cognome da trovare*. Sulla nostra maschera si sovrappone una finestra di messaggio Access che segnala un errore (la prima riprodotta nella Figura 6.11). Se, dopo aver letto attentamente il testo della segnalazione, facciamo clic su *OK*, si apre semplicemente un'altra finestra di dialogo, molto criptica (la seconda della figura), nella quale l'unica opzione che ci è concessa è chiuderla con un clic sul pulsante *Interrompi tutte le macro*.

La prima segnalazione di errore non è un capolavoro di chiarezza, ma il testo del primo capoverso fa capire – a chi conosce i termini tecnici di Access – che non è stato possibile eseguire l'azione macro *TrovaRecord* perché non le è stato passato un valore per l'argomento *Trova*, che è, ovviamente, obbligatorio. Il resto del messaggio è quasi impenetrabile e induce a sperare che, premendo il pulsante *OK*, si possa risolvere il problema che ha causato l'errore: ma non è così, perché la successiva

finestra di messaggio è soltanto per addetti ai lavori e non consente alcun intervento correttivo. Chiuso il secondo avvertimento ci si ritrova davanti alla maschera, sulla quale non è stata eseguita alcuna azione.

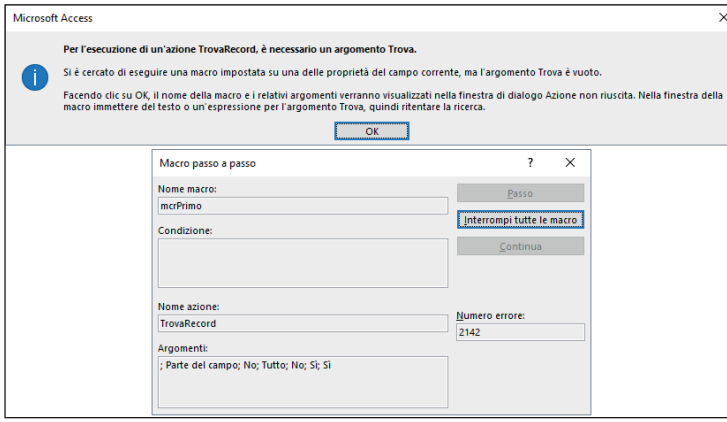


Figura 6.11 Le segnalazioni di errore generate da una macro.

Qualche lettore potrebbe obiettare che non ha senso premere il pulsante di comando *Trova* nella maschera FRMINDIRIZZICONRICERCA senza aver prima digitato nella apposita casella un testo da trovare. Giusto, non ha senso, ma può succedere, per una svista, una manovra sbagliata o altre imperscrutabili e imprevedibili ragioni, che un operatore faccia clic sul pulsante sbagliato nel momento sbagliato.

Una manovra scorretta dell'operatore può provocare uno scompenso all'interno di Access, chiamato *errore di run-time*, cioè un errore che si manifesta durante l'esecuzione dell'applicazione e la blocca.

Gli errori di run-time non sempre sono generati da manovre sbagliate dell'operatore. Per esempio, stiamo lavorando tramite una maschera su una tabella che risiede nel disco rigido di un altro computer collegato in rete: se qualcuno spegne quel computer si verifica un errore di run-time, perché la maschera non "vede" più i dati ai quali è associata. Errori di run-time possono manifestarsi anche quando si usano dati organizzati in modo scorretto; immaginiamo di avere una query di calcolo che ottiene una serie di valori dividendo il contenuto del campo di una tabella per quello del campo di un'altra tabella: se in uno dei record della seconda tabella il campo usato come divisore contiene un

valore zero, quando la query lo prende in considerazione si verifica un errore di run-time, perché in aritmetica non è consentito dividere un numero per zero.

Quando si verifica un errore di run-time dovuto a una manovra scorretta dell'operatore (non *se* si verifica, ma *quando*, perché succede sempre) e a maggior ragione se l'errore dipendesse da altre cause, l'applicazione non deve bloccarsi, non deve dare segnalazioni di errore inintelligibili e deve consentire a chi ha sbagliato di rimediare. Tra le azioni macro di Access 2019 ve n'è una, chiamata *SuErrore*, con la quale è possibile intercettare un errore di run-time e tentare di rimediarlo, ma il suo utilizzo è piuttosto macchinoso: per gestire correttamente i potenziali errori di run-time è molto meglio utilizzare routine VBA scritte *ad hoc*, con le quali si può intercettare un errore, dare a chi sta usando l'applicazione spiegazioni chiare e rassicuranti e metterlo in condizione di proseguire nel suo lavoro senza intoppi.

I moduli

Le azioni macro sono, in realtà, routine o funzioni predefinite, scritte in linguaggio Visual Basic for Applications, che vengono completate con i valori previsti come argomento per ciascuna azione. Per questa loro caratteristica è possibile convertire una macro in una funzione VBA, da utilizzare come tale, integrandola con opportuni enunciati per gestire eventuali errori dell'operatore.

Per la conversione di una macro nel suo equivalente in codice VBA basta eseguire un semplice comando dalla barra multifunzione. Proviamo quindi a convertire la nostra macro mcrPrimo per vedere che cosa succede.

Apriamo la macro in visualizzazione *Struttura* e diamo il comando *Strumenti macro/Progettazione/Strumenti/Converti macro in Visual Basic*, ottenendo l'apertura della piccola finestra di dialogo che è riportata nella Figura 6.12.

Spuntiamo entrambe le caselle di controllo e facciamo clic sul pulsante *Converti*. Dopo un attimo si apre una nuova finestra, sulla quale campeggia una finestra di messaggio che ci avverte che la conversione è terminata.

Chiudiamo il messaggio con un clic su *OK* e siamo di fronte alla nuova finestra, che è quella dell'Editor di Visual Basic, aperta su un modulo chiamato Macro mcrPrimo convertita (Figura 6.13).

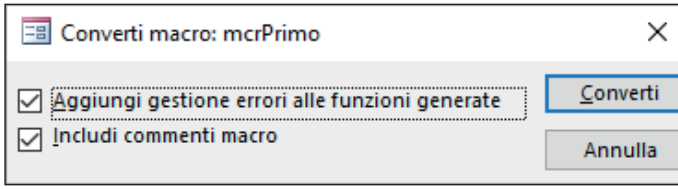


Figura 6.12 Le opzioni per la conversione in VBA di una macro.

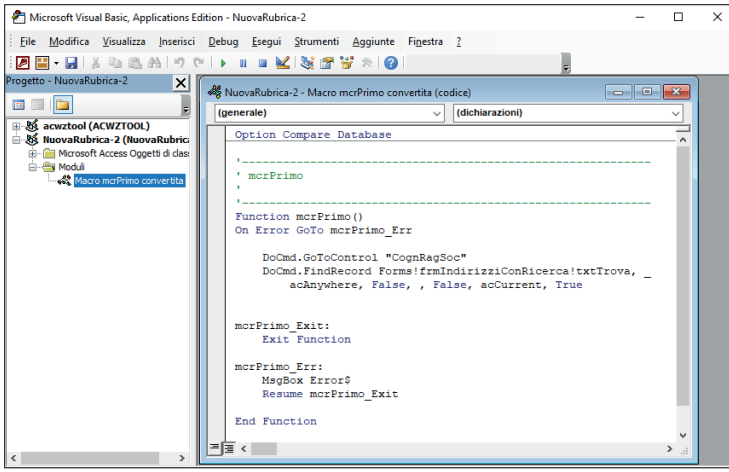


Figura 6.13 La fine della conversione della macro in VBA viene segnalata con la contestuale apertura dell'Editor di Visual Basic.

Nel Capitolo 1 abbiamo descritto brevemente i moduli e l'Editor di Visual Basic col quale i moduli si gestiscono, si creano e si modificano, quindi abbiamo già un'idea di quel che stiamo vedendo. Quando riceve un comando di conversione di una macro in VBA, Access apre l'Editor di Visual Basic e gli fa creare un nuovo modulo, che salva automaticamente col nome **Macro xxx convertita**, dove **xxx** è il nome della macro (che rimane nella sua forma originale nel pannello *Macro* del Riquadro di spostamento).

Il modulo contiene il codice VBA di ogni azione macro, completata con i suoi argomenti, sotto forma di funzione. Gli enunciati VBA che costituiscono la funzione ottenuta dalla conversione delle due azioni della macro MCRPRIMO sono riportati qui di seguito.

```
'-------------------------------------------------------

' mcrPrimo

'-------------------------------------------------------

Function mcrPrimo()

On Error GoTo mcrPrimo_Err

  DoCmd.GoToControl "CognRagSoc"

  DoCmd.FindRecord Forms!frmIndirizziConRicerca!txtTrova, _
     acAnywhere, False, , False, acCurrent, True

mcrPrimo_Exit:

  Exit Function

mcrPrimo_Err:

  MsgBox Error$

  Resume mcrPrimo_Exit

End Function
```

Le righe di codice che iniziano con un apostrofo (e che nell'Editor di Visual Basic sono evidenziate in verde) sono *commenti*, in questo caso generati automaticamente dal processo di conversione.

Anche le righe che terminano con il carattere due punti non sono enunciati, come non lo sono i commenti, ma *etichette*, come si chiamano nel gergo VBA, cioè segnaposti utilizzati dagli enunciati che gestiscono gli errori.

Gli enunciati operativi veri e propri, che eseguono le azioni macro convertite, in questo caso sono quelli che iniziano con la parola chiave `DoCmd`, quindi due in tutto; gli altri enunciati servono per la gestione degli errori.

Per capire come funziona questa gestione bisogna sapere che, normalmente, gli enunciati che compongono una routine o una funzione vengono eseguiti uno dopo l'altro fino all'enunciato di chiusura; se si verifica un errore, di qualunque tipo, l'esecuzione della routine si arresta.

Il primo enunciato che si trova subito dopo il nome della funzione ha il compito di intercettare un eventuale errore di run-time e di saltare l'esecuzione degli enunciati operativi, andando a eseguire quelli che vengono dopo una etichetta:

```
On Error GoTo mcrPrimo_Err
```

Tradotto in italiano, questo enunciato dice: "In caso di errore vai direttamente all'etichetta `mcrPrimo_Err`".

Subito sotto l'etichetta `mcrPrimo_Err:` si trovano due enunciati, il primo presenta una finestra di messaggio con un testo predefinito e il secondo:

```
Resume mcrPrimo_Exit
```

è l'ordine di continuare l'esecuzione della funzione (dopo che l'utente ha chiuso la finestra di messaggio), riprendendo dal punto segnato da un'altra etichetta (`mcrPrimo_Exit:`). Dopo questa etichetta c'è un enunciato:

```
Exit Function
```

che fa chiudere la funzione.

Convertendo in Visual Basic la macro MCRSUCCESSIVO si ottiene una funzione identica, cambiano soltanto le etichette, generate automaticamente a partire dal nome della macro.

Per vedere l'effetto che si ottiene usando il codice VBA ottenuto dalla conversione delle macro, non dobbiamo fare altro che modificare il riferimento alla macro MCRTROVA nella casella della proprietà *Su clic* dei due pulsanti di comando, in questo modo:

Pulsante di comando	**Proprietà Evento *Su clic***
CMDTROVA	**=mcrPrimo()**
CMDANCORA	**=mcrSuccessivo()**

Questa notazione significa "Quando il pulsante di comando sente un clic deve chiamare la funzione il cui nome è scritto dopo il segno di uguale" e può essere utilizzata soltanto con le funzioni VBA e non con le routine.

Nel gergo della programmazione le funzioni si "chiamano" mentre le routine si "eseguono": la differenza lessicale nasce dal fatto che le funzioni sono programmi che possono essere eseguiti soltanto con un comando emesso da un oggetto software (che le "chiama"), mentre le routine possono essere eseguite anche con un comando manuale oltre che con un comando emesso da un altro oggetto software. Le funzioni restituiscono il risultato della loro elaborazione all'oggetto software che le ha chiamate, mentre le routine non lo fanno, perché non possono rilevare se l'ordine di esecuzione è venuto da un operatore o da un oggetto software.

Salviamo le modifiche e proviamo a ripetere deliberatamente l'errore, premendo il pulsante *cmdTrova* senza aver prima digitato una stringa di caratteri nella casella di testo. L'input incoerente produce un errore di run-time, che viene intercettato dall'enunciato per la gestione degli errori, la funzione si arresta e viene eseguito l'enunciato:

```
MsgBox Error$
```

che apre la finestra di messaggio che vediamo nella Figura 6.14. Si tratta di un messaggio standard, quindi troppo generico per essere di aiuto all'operatore, ma almeno non è ingombrante e farraginoso come la coppia di finestre di messaggio della Figura 6.12.

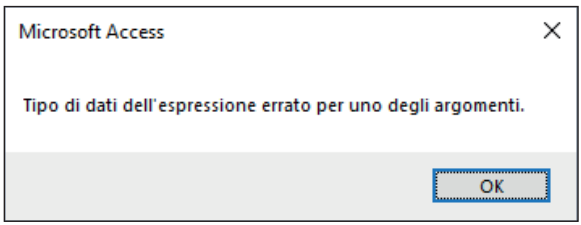


Figura 6.14 La finestra di messaggio attivata dalla funzione VBA segnala l'errore in un modo diverso.

Possiamo migliorare notevolmente le cose con una semplice modifica dell'enunciato che segnala l'errore. Questo enunciato è composto dalla parola chiave `MsgBox`, un'istruzione che apre una finestra di messaggio, seguita dal nome di una variabile, `Error$`, che contiene la descrizione standard per l'errore rilevato. Mettiamo la maschera in

visualizzazione *Struttura* e facciamo clic sul pulsante *Visualizza Codice* in *Strumenti struttura maschera/Progettazione/Strumenti*.

Il comando apre l'Editor di Visual Basic dando così accesso alle due funzioni ottenute con la conversione delle macro.

Selezioniamo in ciascuna di esse l'intera riga che contiene l'enunciato `MsgBox Error$` e digitiamo al suo posto l'enunciato seguente:

```
MsgBox "La casella col cognome da cercare è vuota"
```

Salviamo le modifiche, usciamo dall'Editor di Visual Basic e dalla finestra di Access apriamo di nuovo la maschera, provocando il solito errore: la segnalazione che compare (Figura 6.15) adesso è più chiara e aiuta l'operatore a capire dove ha sbagliato.

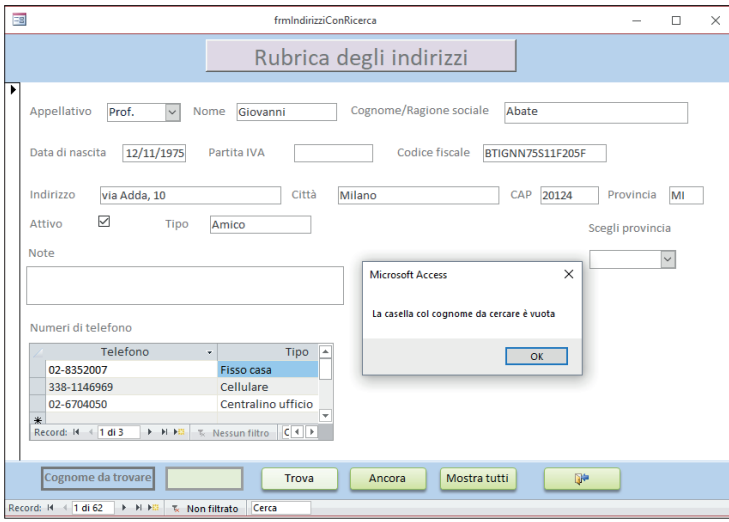


Figura 6.15 La finestra con un messaggio di errore personalizzato aiuta l'operatore a capire dove ha sbagliato.

I moduli sono contenitori di stringhe di testo, nei quali si scrivono routine e funzioni VBA con l'aiuto dell'Editor di Visual Basic. Dopo aver scritto una routine in un modulo è possibile eseguirla, restando all'interno dell'Editor, per provarla e verificare che i suoi risultati corrispondano a quello che si intende ottenere.

Nel rispetto delle regole sintattiche rigorose che stanno alla base di tutti i linguaggi di programmazione, si può ottenere lo stesso risultato in molti modi diversi: gli esempi di codice che presentiamo sono, appunto, esempi e non indicazioni vincolanti su come si devono scrivere le routine e le funzioni VBA.

Un'applicazione Access può contenere due tipi di moduli: i *moduli standard* e i *moduli di classe*. I primi sono quelli visualizzati nel pannello *Moduli* della finestra *Database* e possono essere creati liberamente e nel numero che si desidera con il comando *Crea/Macro e codice/Modulo*.

I moduli di classe sono strutturalmente simili a quelli standard, ma non hanno un nome che li individui e può esisterne soltanto uno associato a una maschera o a un report. Un modulo di classe viene aggiunto a una maschera o a un report quando si imposta una delle sue proprietà *Evento* sull'opzione predefinita **(***Routine evento***)** invece che sul nome di una funzione contenuta in un modulo standard (nel prossimo paragrafo vedremo come si genera un modulo di classe).

Per accedere a tutti i moduli, standard e di classe, contenuti in un'applicazione, si apre l'Editor di Visual Basic con il comando *Crea/Macro e codice/Visual Basic* e si sceglie il comando *Gestione progetti* dal menu *Visualizza*, ottenendo il risultato che vediamo nella Figura 6.16.

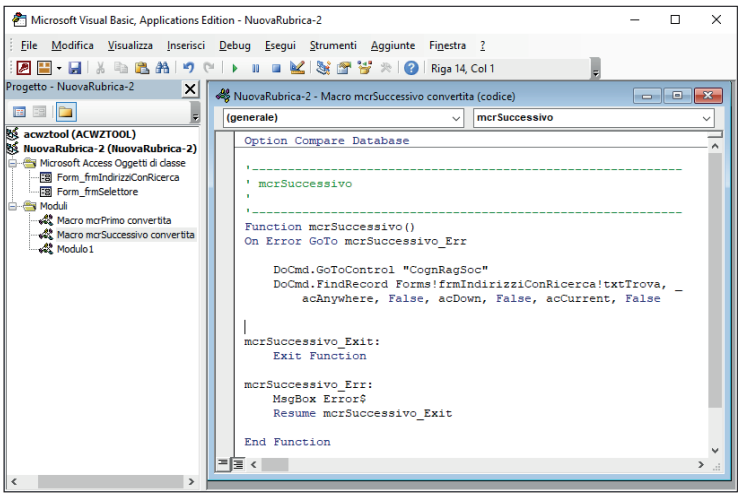


Figura 6.16 L'Editor di Visual Basic elenca nel riquadro Gestione progetti tutti i moduli presenti nel file .accdb.

Un doppio clic su uno dei nomi elencati nel riquadro di sinistra (quello della *Gestione progetti*) apre nel riquadro a destra (quello dell'Editor vero e proprio) il modulo corrispondente, di classe o standard.

Per lavorare comodamente con l'Editor sul codice di un modulo, dopo averlo aperto con un doppio clic sul suo nome nel riquadro *Gestione progetti* si può chiudere questo riquadro e guadagnare spazio visualizzando soltanto il contenuto del modulo.

Il linguaggio VBA

Il linguaggio di programmazione Visual Basic for Applications è un sottoinsieme, concepito per le applicazioni Office, di un linguaggio sviluppato da Microsoft per creare applicazioni di qualunque tipo, chiamato Visual Basic, che deriva dal BASIC, un linguaggio nato nel remoto 1964 per insegnare la programmazione dei computer agli studenti delle scuole superiori. Le sue finalità didattiche rendevano il BASIC un linguaggio estremamente semplice da imparare e da utilizzare e questa semplicità è tuttora presente nel Visual Basic e, per estensione, nel VBA, come possiamo vedere dalla breve panoramica che segue.

Il VBA ha un lessico e una sintassi:

- il *lessico* è composto da una cinquantina di parole chiave, che vanno scritte sempre nello stesso modo;
- la *sintassi* è un insieme di regole rigide che stabiliscono come combinare fra loro le parole chiave per ottenere istruzioni ed enunciati.

Le *istruzioni* sono composte con una o più parole chiave ed equivalgono a ordini dati al computer. Normalmente una istruzione si completa con *valori* (numeri o stringhe di caratteri) sui quali l'istruzione agisce e l'insieme composto da un'istruzione e dal valore o dai valori che questa utilizza si chiama *enunciato*.

Quasi sempre, invece di scrivere direttamente un valore in un enunciato, si preferisce assegnare un nome convenzionale al valore e scrivere questo nome nell'enunciato; la stringa di caratteri che compone il nome convenzionale assegnato a un valore si chiama *variabile*.

Prima di usare una variabile è necessario definirla con un enunciato detto di *dichiarazione*, formato dall'istruzione `Dim`, completata dalla parola chiave `As` e da un'altra parola chiave che specifica il tipo di dati, nella forma seguente.

```
Dim strSaluto As String
```

il cui significato, in questo caso, è "Definisci una variabile chiamata STRSALUTO con tipo dati `String`".

Vediamo un esempio per capire meglio come funziona questo meccanismo. L'enunciato:

```
MsgBox "Salve ragazzi!"
```

è formato dall'istruzione `MsgBox` completata con il valore assoluto (la stringa racchiusa fra virgolette doppie) sul quale l'istruzione agisce. Eseguendo questo enunciato, si ottiene una finestra di messaggio che contiene al suo interno il testo della stringa.

Se scriviamo lo stesso enunciato nella forma:

```
MsgBox strSaluto
```

l'istruzione `MsgBox` agisce sulla variabile `strSaluto`, il cui contenuto può essere definito a parte, con un enunciato detto di *assegnazione* fatto in questo modo:

```
strSaluto = "Salve ragazzi!"
```

```
oppure in quest'altro:
```

```
strSaluto = "Signori buongiorno!"
```

Il segno di uguale usato negli enunciati di assegnazione non stabilisce un'uguaglianza, ma assegna alla variabile, il cui nome si scrive a sinistra, il valore posto a destra del segno. Nel caso di variabili che rappresentano stringhe, il valore che si assegna va racchiuso fra virgolette doppie, mentre, se si assegna un valore numerico, lo si scrive senza virgolette.

La piccola routine esemplificativa che la Figura 6.17 mostra all'interno dell'Editor di Visual Basic, quando viene eseguita apre la finestra di messaggio che compare sotto.

Modificando l'enunciato di assegnazione nel modo evidenziato dalla riga in grassetto:

```
Sub TestMessaggio()

        Dim strSaluto As String

        strSaluto = "Signori buongiorno!"

        MsgBox strSaluto

End Sub
```

la routine apre, invece, la finestra di messaggio che è riprodotta nella Figura 6.18.

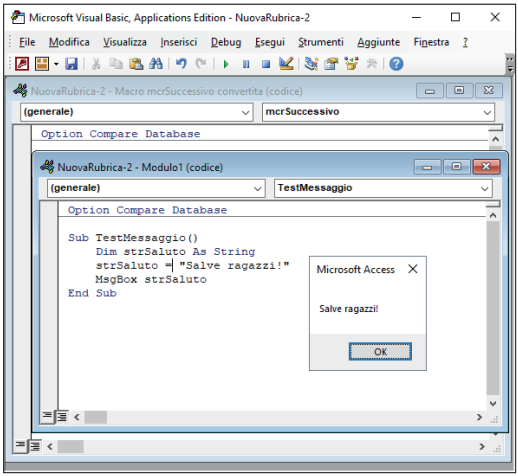


Figura 6.17 Una semplice routine VBA che apre una finestra di messaggio quando viene eseguita.

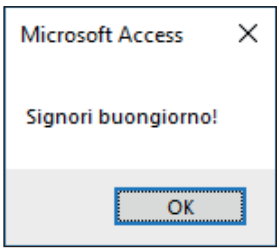


Figura 6.18 Cambiando il valore della variabile cambia anche il contenuto della finestra di messaggio.

Nelle routine VBA si può lavorare con le proprietà degli oggetti Access semplicemente richiamando in un enunciato il nome dell'oggetto e quello della proprietà che si intende utilizzare, congiungendo i due nomi con un carattere punto oppure punto esclamativo

I nomi degli oggetti possono essere sostituiti con variabili, che si associano agli oggetti con gli stessi enunciati di assegnazione usati per le variabili che identificano stringhe di caratteri o valori numerici.

In Access, gli oggetti sono organizzati secondo una gerarchia che va dal generale al particolare, per cui il riferimento a un controllo che sta in una maschera va scritto in questo modo in un modulo standard:

```
Forms!frmIndirizziConRicerca!txtTrova
```

specificando così la casella di testo `txtTrova` che fa parte della maschera `frmIndirizziConRicerca`, che a sua volta appartiene all'insieme di oggetti predefinito `Forms`.

Nei soli moduli di classe si può identificare la maschera alla quale è associata il modulo senza far riferimento all'insieme di oggetti `Forms` e utilizzando una variabile predefinita chiamata `Me`, per cui la stessa casella di testo `txtTrova` può essere identificata in un modo più conciso:

```
Me.txtTrova
```

se il modulo di classe è associato alla maschera FRMINDIRIZZICONRICERCA. Nulla vieta di utilizzare anche in un modulo di classe la notazione gerarchica completa che abbiamo visto sopra.

I valori che si impostano per le proprietà dei controlli nelle maschere quando si creano o si modificano in visualizzazione *Struttura* possono essere modificati dinamicamente da una routine VBA, mediante semplici enunciati di assegnazione. Per esempio, se un pulsante di comando chiamato CMDTEST è abilitato (quindi può essere premuto), la sua proprietà *Abilitato* compare impostata su *Sì* nella scheda *Dati* della sua *Finestra delle proprietà*. Questa proprietà può essere disattivata da un enunciato VBA come questo:

```
Me.cmdTest.Enabled = False
```

e riattivata con quest'altro enunciato di assegnazione:

```
Me.cmdTest.Enabled = True
```

I due enunciati si scrivono in questo modo quando fanno parte di routine inserite nel modulo associato alla maschera che contiene il controllo chiamato CMDTEST.

Gli enunciati che compongono una routine VBA vengono eseguiti sequenzialmente fino all'enunciato di chiusura, che di norma è l'ultima riga (`End Sub` oppure `End Function`), ma può essere anche un enunciato `Exit Sub` o `Exit Function`, collocato prima dell'ultima riga.

Come abbiamo visto prima, parlando della gestione degli errori, l'esecuzione sequenziale si interrompe in caso di errore. Può essere necessario modificare la sequenza deliberatamente, passando a eseguire un enunciato o un altro in base al valore assunto da una determinata variabile: a questo scopo si utilizza una istruzione particolare, composta dalle parole chiave `If`, `Then`, `Else` ed `End If`, con la quale si formano enunciati che hanno questa struttura

```
If condizione Then

        [enunciati da eseguire se condizione ha valore
         logico Vero]

Else

        [enunciati da eseguire se condizione ha valore
         logico Falso]

End If
```

l'enunciato di *diramazione* che si costruisce con l'istruzione `If…Then… End If` può non contenere la clausola `Else`, nel qual caso assume la forma:

```
If condizione Then

        [enunciati da eseguire se condizione ha valore
         logico Vero]

End If

        [enunciati da eseguire se condizione ha valore
         logico Falso]
```

Terminiamo questa breve panoramica delle caratteristiche essenziali del VBA con un cenno alla parola chiave `DoCmd`, che abbiamo già incontrato negli esempi precedenti. `DoCmd` non è una istruzione VBA ma è il nome di un poderoso oggetto software interno ad Access, in grado di eseguire tutte le azioni macro e lo si utilizza per costruire enunciati molto complessi che hanno la forma:

```
DoCmd.azionemacro argomento1, argomento2, , , argomentoN
```

dove *azionemacro* è il nome di una delle azioni che si possono eseguire con le macro, eccetto alcune (otto, per l'esattezza) e i vari *argomento1*, *argomento2* e così via sono gli argomenti specifici di ciascuna azione; l'ordine in cui vengono elencati gli argomenti permette all'oggetto `DoCmd` di distinguerli uno dall'altro. Per molte azioni è obbligato-

rio un solo argomento, che va scritto subito dopo il nome dell'azione, separato da uno spazio, mentre gli argomenti facoltativi si possono omettere. Se, però, si scrive espressamente il nome di un argomento facoltativo omettendone alcuni, la loro omissione va segnalata con l'inserimento di una virgola, come nell'enunciato che segue:

```
DoCmd.FindRecord Forms!frmIndirizziConRicerca!txtTrova,
acAnywhere, , , , acCurrent, False
```

Ancora un'annotazione: gli enunciati VBA si scrivono su una sola riga, che può estendersi per un numero mostruoso di caratteri. Per non perdere l'orientamento quando si scrive codice VBA è consentito scomporre la riga di un enunciato particolarmente lungo in più spezzoni, avendo cura di inserire nel punto di cesura uno spazio seguito da un trattino basso, in questo modo:

```
DoCmd.FindRecord Forms!IndirizziConRicerca!txtTrova, _
       acAnywhere, False, acDown, False, _
       acCurrent, False
```

Il trattino basso preceduto da uno spazio può essere inserito soltanto dopo un punto o una virgola, cioè nella posizione in cui inizia un elemento distinto dell'enunciato, come nell'esempio indicato sopra. Non è consentito ricorrere a questa tecnica per spezzare righe molto lunghe che assegnano una stringa di testo a una variabile; per esempio, se volessimo attribuire a una variabile chiamata `strDante` i primi versi della *Divina Commedia*, non è consentito scrivere l'enunciato di assegnazione come segue:

```
strDante = "Nel mezzo del cammin di nostra vita _
              mi ritrovai per una selva oscura, _
              ché la diritta via era smarrita"
```

ma bisogna segmentare la stringa in sottostringhe, ciascuna racchiusa da virgolette doppie, congiungendole con l'operatore di concatenamento, che è il carattere &, in questo modo:

```
strDante = "Nel mezzo del cammin di nostra vita " _
         & "mi ritrovai per una selva oscura, " _
         & "ché la diritta via era smarrita"
```

Sempre per migliorare la leggibilità del codice, si usa inserire una tabulazione in testa ad alcune righe per evidenziare la loro appartenenza a un gruppo omogeneo. Queste tabulazioni non hanno alcun effetto sull'esecuzione del codice.

Un esempio articolato

Sulla scorta di questa breve panoramica del linguaggio Visual Basic for Applications proviamo a sviluppare un esempio un po' più articolato di quelli visti finora, per capire meglio come si possono aggiungere funzionalità alle maschere di un'applicazione; con l'occasione, esaminiamo anche come si integrano istruzioni SQL in routine VBA.

Abbiamo visto come impostare un filtro su un campo Provincia per ridurre il numero dei record da esplorare con una maschera frmIndirizzi. Sviluppiamo ora una maschera di servizio che consente di fare un passo in più: selezionare una provincia e ottenere contestualmente nella stessa maschera un elenco dei nomi delle persone e delle imprese registrate nella tabella tblIndirizzi che corrispondono alla provincia selezionata, come nella Figura 6.19.

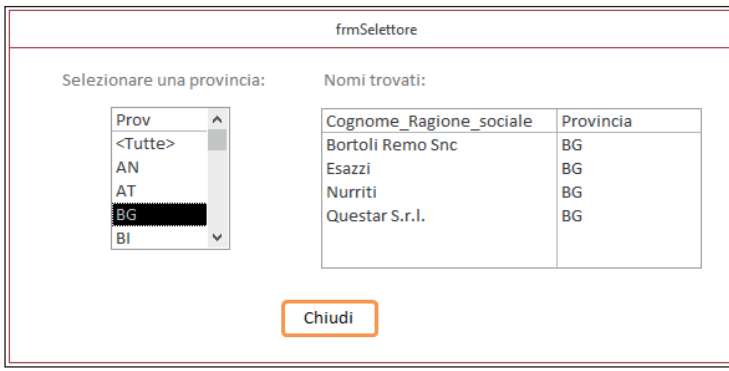


Figura 6.19 Questa maschera elenca nella casella di riepilogo di destra gli estremi dei record che corrispondono alla provincia selezionata nella casella di riepilogo di sinistra.

Creiamo una maschera non associata che salviamo col nome **frmSelettore**, dopo aver impostato le proprietà barre di spostamento, pulsanti di ingrandimento/riduzione e così via in modo che la maschera abbia l'aspetto che vediamo nella figura.

Inseriamo alla base della maschera un pulsante di comando, al quale assegniamo il nome cmdChiusura, impostando su **[*Routine evento*]** la sua proprietà evento *Su clic*. Nella casella di questa proprietà facciamo clic sul pulsante *Genera* ottenendo la creazione del modulo di classe della maschera e la contestuale impostazione dello schema vuoto della rou-

tine evento. Inseriamo in quello schema un enunciato basato sull'oggetto `DoCmd` in questo modo:

```
Private Sub cmdChiusura_Click()
        'Chiude la maschera
        DoCmd.Close
End Sub
```

Quando si attiva la creazione di un modulo di classe associato a una maschera o a un report, i nomi delle routine VBA che vengono automaticamente impostate iniziano con la parola chiave `Private`*. Le routine così caratterizzate si possono attivare soltanto dall'interno del modulo che le contiene. Volendo rendere accessibile una routine anche da altre parti, si sostituisce* `Private` *con la parola chiave* `Public` *o si omette il qualificatore.*

Inseriamo due caselle di riepilogo dimensionandole come nella figura e assegniamo a quella di sinistra il nome **lstListaProvince** e a quella di destra il nome **lstNomiRubrica**.

Impostiamo per la casella di riepilogo LSTNOMIRUBRICA queste proprietà:

Scheda	Proprietà	Valore
Formato	*Numero colonne*	**2**
	Intestazioni colonne	**Sì**
	Larghezza colonne	**5 cm;1 cm**
Dati	*Colonna associata*	**1**

Apriamo la *Finestra delle proprietà* della casella di riepilogo LSTLISTAPROVINCE e digitiamo nella casella *Origine riga* della scheda *Dati* la seguente istruzione SQL:

```
SELECT DISTINCT Prov FROM tblIndirizzi
UNION SELECT "<Tutte>" FROM tblIndirizzi;
```

Questa query non si può generare nella griglia di struttura delle query perché è specifica del linguaggio SQL e non di Access: si tratta di una query detta di *unione* perché usa la parola chiave `UNION` e serve per generare una tabella virtuale che contiene i dati di tutti i record specificati in due o più tabelle senza però includere i record duplicati (per includere i duplicati bisogna usare la parola chiave `ALL` invece di

DISTINCT). La tabella virtuale generata da questa query è quella che si vede nella Figura 6.20: tutte le sigle univoche delle province, in ordine alfabetico, precedute dalla stringa **<Tutte>**, che viene generata dal secondo comando SELECT.

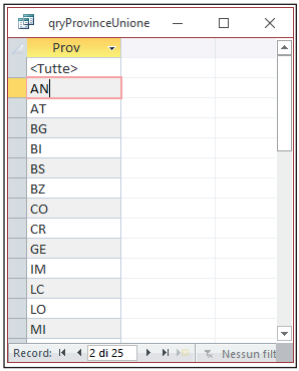


Figura 6.20 Il risultato della query inserita come origine dati nella casella di riepilogo lstListaProvince.

Passiamo alla scheda *Evento* della *Finestra delle proprietà* e scegliamo la casella della proprietà *Dopo aggiornamento*, dove selezioniamo l'opzione **[***Routine evento***]**; facciamo clic sul pulsante *Genera*, provocando la riapertura del modulo di classe generato quando abbiamo inserito la routine evento per il pulsante di comando. Completiamo lo schema della routine evento in questo modo:

```
Private Sub lstListaProvince_AfterUpdate()

  'Se nella casella di riepilogo lstListaProvince
  'è stato selezionato il valore <Tutte>,
  'imposta la proprietà Origine dati della casella
  'di riepilogo lstNomiRubrica su una query
  'che estrae i campi CognRagSoc e Prov
  'da tutti i record della tabella Indirizzi...


  If Me.lstListaProvince = "<Tutte>" Then
     Me.lstNomiRubrica.RowSource = _
       "SELECT CognRagSoc AS Cognome_Ragione_sociale, " _
       & "Prov AS Provincia FROM tblIndirizzi " _
       & " ORDER BY tblIndirizzi.CognRagSoc"
```

```
  '... altrimenti imposta la proprietà Origine dati
  'della casella di riepilogo lstNomiRubrica su una
  'query che estrae i campi CognRagSoc e Prov
  'dai soli record della tabella tblIndirizzi che
  'hanno nel campo Prov il valore selezionato nella
  'casella di riepilogo lstListaProvince

  Else
     Me.lstNomiRubrica.RowSource = _
       "SELECT CognRagSoc AS Cognome_Ragione_sociale, " _
       & " Prov AS Provincia FROM tblIndirizzi " _
       & "WHERE Prov = '" & Me.lstListaProvince & "'" _
       & " ORDER BY tblIndirizzi.CognRagSoc"
  End If

End Sub
```

Ripetiamo la stessa operazione per la casella di riepilogo LSTNOMIRUBRICA, digitando questa routine per l'evento *Dopo aggiornamento*:

```
Private Sub lstNomiRubrica_AfterUpdate()

  'Crea una variabile per contenere il valore selezionato
  'nella casella di riepilogo lstNomiRubrica

  Dim strSelezione As String

    'Rigenera il contenuto di se stessa

    Me.Recalc

    'Trasferisce nella variabile il valore selezionato
    'nella casella di riepilogo lstNomiRubrica

    strSelezione = Me.lstNomiRubrica

    'Apre la maschera frmIndirizzi mostrando soltanto
    'il record che nel campo CognRagSoc contiene
    'il valore assegnato alla variabile strSelezione
    'Attenzione all'uso delle virgolette semplici
    'e doppie!

    DoCmd.OpenForm "frmIndirizziConRicerca", , ,
      "CognRagSoc =  '" & strSelezione & "'"

End Sub
```

Si osservi che nella parte finale dell'ultimo enunciato vi sono due caratteri virgoletta semplice (`''`) che vanno scritti senza lasciare spazi fra questi e le virgolette doppie che li seguono o li racchiudono.

Salviamo le modifiche e apriamo la maschera in visualizzazione *Maschera*, che si presenterà, dopo la selezione di un qualunque valore nella casella di riepilogo di sinistra, come nella Figura 6.19.

Adesso che abbiamo un elenco dei soli record della provincia che ci interessa, possiamo selezionare agevolmente il singolo record che vogliamo esaminare facendo clic sul nome che compare nella casella di riepilogo di destra: si apre la maschera FRMINDIRIZZICONRICERCA presentando soltanto il record che ci interessa, come vediamo nella Figura 6.21

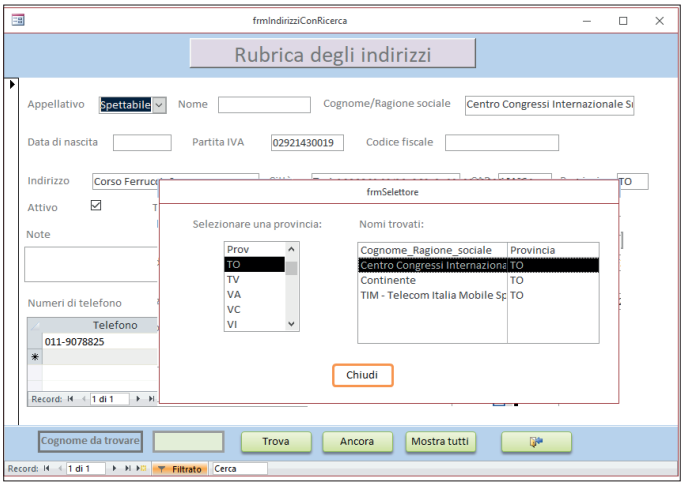


Figura 6.21 Dalla maschera frmSelettore si passa direttamente alla maschera frmIndirizziConRicerca, aperta sul singolo indirizzo selezionato nella casella di riepilogo di destra.

Chiudendo la maschera FRMINDIRIZZICONRICERCA si torna alla maschera FRMSELETTORE, dalla quale si può avviare un'altra ricerca.

I commenti inseriti nel codice delle due routine evento dovrebbero essere sufficienti per capire come funzionano. Aggiungiamo comunque qualche chiarimento, in particolare sull'utilizzo delle istruzioni SQL dall'interno di una routine VBA.

La prima routine evento, quella che viene eseguita quando si seleziona un valore nella casella di riepilogo LSTLISTAPROVINCE, è predisposta per assegnare due diverse istruzioni SQL alla proprietà *Origine dati* (in inglese *RowSource*) della casella di riepilogo LSTNOMIRUBRICA; il primo

comando SQL ha questa forma, quando la stringa è stata ricomposta dall'operatore di concatenamento &:

```
SELECT CognRagSoc AS Cognome_Ragione_sociale,
       Prov AS Provincia

FROM tblIndirizzi

ORDER BY tblIndirizzi.CognRagSoc
```

e viene eseguita se l'utente ha fatto clic sul valore **<*Tutte*>** nella casella di riepilogo lstListaProvince; l'esecuzione di questa query genera nella casella di riepilogo lstNomiRubrica un elenco con i campi CognRagSoc e Prov di tutti i record della tabella tblIndirizzi, presentando i due campi con un nome più chiaro (*Cognome_Ragione_sociale e Provincia*) ottenuto usando la clausola AS.

La seconda istruzione SQL, riportata qui di seguito, ha la stessa finalità ed è quasi uguale alla precedente, però viene eseguita se l'utente ha fatto clic su un valore diverso da <*Tutte*> nella casella di riepilogo lstListaProvince e usa la clausola WHERE, che stabilisce un criterio di selezione per i record da prendere in considerazione nella tabella tblIndirizzi: vanno estratti solo i record che nel campo Prov hanno lo stesso valore che al momento è selezionato nella casella di riepilogo lstListaProvince:

```
SELECT CognRagSoc AS Cognome_Ragione_sociale,
       Prov AS Provincia

FROM tblIndirizzi WHERE Prov =  Me.lstListaProvince

ORDER BY tblIndirizzi.CognRagSoc
```

La routine assegna una o l'altra delle istruzioni SQL alla proprietà *Origine record* della casella di riepilogo lstNomiRubrica mediante un enunciato `If...Then...Else`.

Per effetto delle proprietà che sono state impostate per la casella di riepilogo lstNomiRubrica, questa presenta l'elenco dei suoi valori su due colonne, le colonne hanno un'intestazione generata dalla query SQL e un clic su una delle righe dell'elenco determina l'associazione con il valore della prima colonna.

La routine evento per questo controllo sfrutta l'associazione con la prima colonna e, quando si fa clic su una voce elencata nella casella di riepilogo, trasferisce il valore della prima colonna in una variabile chiamata strSelezione, definita all'inizio della routine con tipo dati `String`, quindi atta a contenere una stringa di caratteri, qual è il contenuto del campo CognRagSoc della tabella tblIndirizzi.

Subito dopo attiva l'oggetto DoCmd facendogli eseguire l'azione *TrovaRecord* (*FindRecord*) con gli argomenti necessari, fra i quali, di fondamentale importanza, è l'ultimo, scritto in questo modo:

```
"CognRagSoc =  '" & strSelezione & "'"
```

Questo argomento è il valore della *condizione WHERE*, indica cioè il criterio di selezione da utilizzare per l'azione *TrovaRecord*.

Supponiamo che la variabile strSelezione contenga il valore Bortoli Remo Snc: quando la routine viene eseguita, il VBA compone questo spezzone di riga tenendo conto del valore contenuto nella variabile, delle virgolette semplici e doppie e degli operatori & di concatenamento, ottenendo così questo argomento:

```
"CognRagSoc = 'Bortoli Remo Snc'"
```

per cui l'azione dell'oggetto DoCmd va a cercare un record che nel campo CognRagSoc contenga l'esatta successione di caratteri che è compresa fra le virgolette semplici, quindi il valore Bortoli Remo Snc.

La successione delle virgolette doppie e semplici utilizzate nella formulazione VBA di questa *condizione WHERE* è di importanza critica. Se non fossero disposte come nella riga che vediamo sopra, il VBA potrebbe comporre l'argomento in questo modo:

```
"CognRagSoc = strSelezione"
```

per cui l'azione *TrovaRecord* andrebbe a cercare un record che contenga nel campo CognRagSoc la stringa *strSelezione*, non il valore rappresentato dalla variabile e, naturalmente, non lo troverebbe.

La stessa disposizione di virgolette semplici e doppie vale per la clausola WHERE del secondo enunciato SQL della prima routine evento, scritta in questo modo nel codice:

```
"WHERE Prov = '" & Me.lstListaProvince & "'"
```

che viene eseguita dopo essere stata ricomposta nella forma:

```
WHERE Prov = Me.lstListaProvince
```

richiamando quindi il valore selezionato nella casella di riepilogo lstListaProvince e non la stringa di caratteri Me.lstListaProvince.

Concludiamo con questo esempio l'esplorazione degli strumenti Access concepiti per aggiungere funzionalità su misura alle applicazioni gestionali fondate su database: ulteriori approfondimenti dei due linguaggi di programmazione Visual Basic for Applications e Structured

Query Language che si utilizzano nei moduli si possono trovare in testi specializzati (come il già citato *Costruire applicazioni con Access* di Mike Davis, pubblicato da Apogeo) che vanno ben oltre i limiti di questa introduzione generale a Microsoft Access 2019.

Strumenti complementari

L'ambiente di lavoro di Access mette a disposizione numerosi strumenti complementari, che possono essere utili quando si sviluppano applicazioni gestionali. Vediamo brevemente quali sono e a che cosa servono.

Modelli

Come abbiamo accennato nel Capitolo 1, quando si avvia Access si sceglie se aprire un database esistente o crearne uno nuovo; in questo secondo caso, è possibile selezionare nel pannello di destra un modello predefinito, scegliendo fra i diciotto modelli di esempio, che si trovano sul computer col quale si lavora e sono stati installati contestualmente all'installazione di Access 2019.

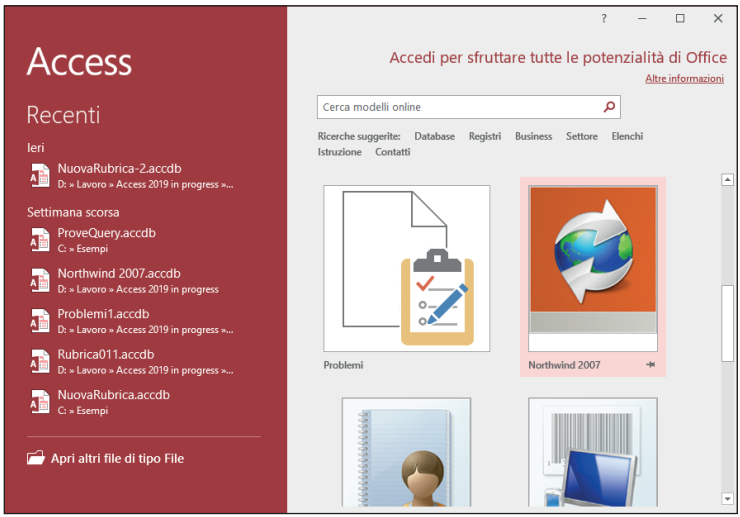


Figura 6.22 Il pannello Modelli disponibili consente di scegliere un modello predefinito per creare un'applicazione database completa.

Se fra i modelli predefiniti non ce n'è uno che si avvicina al risultato che vogliamo ottenere possiamo digitare una domanda nella casella di ricerca che sta in testa alla schermata, andando in questo modo a cercare nel sito web di Microsoft se è disponibile qualcosa di più interessante.

Un clic su uno dei modelli disponibili fa uscire una finestra di dialogo per selezionare il percorso in cui salvare il nuovo file `.accdb` che verrà creato e subito dopo si apre il nuovo database che è un'applicazione completa di dati esemplificativi. Avendo un po' di familiarità con Access (può bastare quella che si ricava dalla lettura di questo libro) si può poi intervenire sui vari oggetti creati automaticamente per modificarli e personalizzarli quanto basta per soddisfare le esigenze specifiche che hanno portato a scegliere quel particolare modello.

Protezione

Un'applicazione gestionale è di solito uno strumento di lavoro importante, al quale è bene che accedano soltanto le persone qualificate per utilizzarlo. Access mette a disposizione due sistemi per proteggere gli accessi a un file `.accdb`, uno molto semplice, basato su una password, chiamato *protezione a livello di condivisione* e un altro, molto più potente ma più complesso da impostare e da gestire, chiamato *protezione a livello di utente*.

Per proteggere un file Access con una password è necessario aprirlo in modalità esclusiva, cosa che si ottiene selezionando l'opzione *Apertura esclusiva* nell'elenco a discesa associato al pulsante *Apri*, come si vede nella Figura 6.23.

Dopo aver aperto in modalità esclusiva il database da proteggere con una password, si sceglie il comando *Crittografa tramite password* nella sezione *Informazioni* della finestra *Backstage*: compare una piccola finestra di dialogo intitolata *Imposta password database*, che chiede di digitare due volte la password, in due distinte caselle di testo, la prima per acquisirla e la seconda per avere una conferma; in entrambe le caselle di testo *Password* e *Verifica* la stringa che viene digitata appare mascherata da asterischi. Un clic su *OK* conferma l'acquisizione della password per quel file Access.

Successivi comandi di apertura del file Access protetto da password provocano l'apertura di una finestra di dialogo che chiede l'immissione (una sola volta) della password. Se la password digitata non è quella corretta compare una segnalazione di errore.

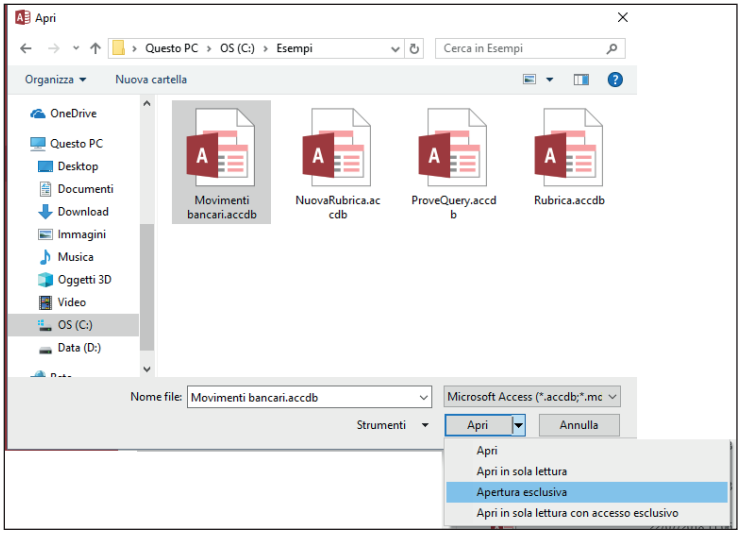


Figura 6.23 Come si apre un file Access esistente con l'opzione Apertura esclusiva.

Per eliminare la password impostata per un file `.accdb`, si seleziona il comando *Decrittografa database*, sempre nella sezione Informazioni della finestra *Backstage*; constatando che il file è protetto da password, il comando presenta la finestra di dialogo *Annulla password database*, nella quale si deve digitare correttamente la password esistente da eliminare.

Le varie finestre di dialogo che compaiono quando si lavora con le password sono raggruppate nella Figura 6.24.

Il comando col quale si imposta una password si chiama *Crittografa tramite password* perché, oltre a proteggere il file Access con una password scelta dall'utente, crittografa tutto il suo contenuto, il che è un'ottima cosa, per la seguente ragione: il contenuto dei file `.accdb` si può esaminare con un qualunque strumento capace di visualizzare i dati di un file binario e per farlo non si deve neppure andare molto lontano, basta l'elaboratore di testi WordPad distribuito con tutte le versioni di Windows. Una volta aperto un file `.accdb` con uno strumento adeguato, è possibile scorrerlo per esaminare il contenuto e fra molti caratteri dall'aspetto strano si ritroveranno chiaramente leggibili tutte le stringhe di testo contenute nei campi delle tabelle. La crittografia

creata col comando *Crittografa tramite password* elimina questo rischio, perché un file Access 2019 protetto da password e crittografato se viene aperto con WordPad o qualunque altro strumento presenta soltanto una successione continua di caratteri incomprensibili.

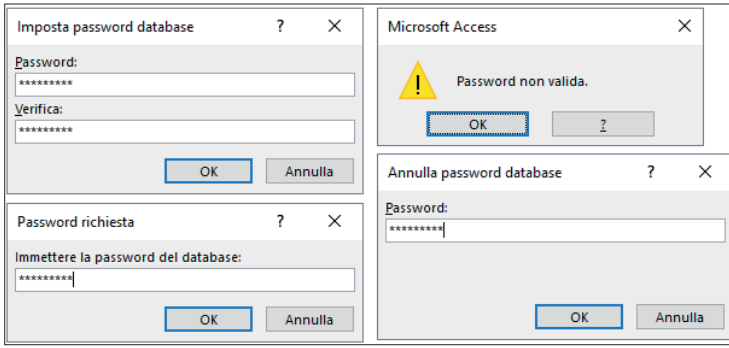


Figura 6.24 Le finestre di dialogo per gestire la protezione di un file Access mediante password.

Ambiente di lavoro personalizzato

Quando si apre un'applicazione database sono disponibili tutte le funzionalità di Access, oltre a quelle specifiche predisposte per l'applicazione. Questo va molto bene mentre si lavora a mettere a punto un'applicazione ma, quando è pronta e viene consegnata agli operatori che la utilizzeranno per accedere al database, inserire nuovi record o modificare record esistenti, consultare selettivamente tabelle mediante maschere e stampare report, tutte le funzionalità che servono per la modifica delle strutture degli oggetti devono essere inibite, per evitare che gli operatori, deliberatamente o per errore, portino qualche oggetto Access in visualizzazione *Struttura* e lo modifichino, devastando così l'applicazione.

Per evitare che questo accada, è necessario predisporre una serie di vincoli all'utilizzo di un'applicazione, dopo che questa è stata messa a punto e ben collaudata, eliminando dall'ambiente di lavoro tutto quello che non serve, quindi menu, barre degli strumenti e così via.

È inoltre importante predisporre un percorso logico per accedere alle maschere e ai report, senza che l'utente debba ricorrere al Riquadro di spostamento per reperire gli oggetti di interfaccia che gli servono.

A questo scopo è opportuno predisporre una maschera non associata, che presenti in una forma chiara e semplice le opzioni che l'utente ha a disposizione, in modo che possa partire da quella maschera nel momento in cui si avvia l'applicazione. La maschera potrebbe avere un aspetto simile a quello che vediamo nell'esempio della Figura 6.25.

Figura 6.25 Da questa maschera l'utente può soltanto accedere alle maschere per consultare gli indirizzi e per selezionarli in base alla provincia.

Rifacendoci agli esempi che abbiamo visto nel paragrafo precedente, un clic sul pulsante *Consulta* chiude la maschera Avvio e apre la maschera frmIndirizziConRicerca, mentre un clic sul pulsante *Seleziona* chiude la maschera Avvio e apre la maschera frmSelettore.

Nelle maschere frmIndirizziConRicerca e frmSelettore dovrà essere stato inserito un pulsante di comando per chiuderle e tornare ad aprire la maschera Avvio. Questi risultati si ottengono con le semplici azioni macro *Chiudi* e *ApriMaschera* che abbiamo visto all'opera nelle pagine precedenti.

Una volta preparata la maschera Avvio e dopo averla collaudata accuratamente, si possono modificare le opzioni di Access in modo da ottenere che, all'apertura dell'applicazione, si apra la maschera Avvio e siano inaccessibili il Riquadro di spostamento, i menu contestuali e tutte le schede della barra multifunzione dalle quali si possono eseguire comandi che modificano la struttura degli oggetti contenuti nel database, così che l'utente non possa, neanche volendo, danneggiare l'applicazione.

Questo risultato si ottiene molto semplicemente dalla pagina *Opzioni del database corrente* delle *Opzioni di Access*. Nella sezione *Opzioni applicazione* si sceglie il nome della maschera da visualizzare all'avvio dalla casella combinata *Visualizza maschera*; si può digitare un titolo per l'applicazione nella casella di testo *Titolo applicazione*, scegliendo eventualmente un'icona per l'applicazione con l'aiuto del pulsante *Sfoglia* a lato della casella di testo *Icona applicazione*.

Nella stessa sezione si deselezionano le caselle di controllo *Attiva visualizzazione Layout* e *Attiva modifiche a livello di struttura per le tabelle in visualizzazione Foglio dati*.

Nella sezione *Spostamento* si toglie il segno di spunta dalle caselle di controllo *Visualizza riquadro di spostamento*, *Menu completi* e *Menu di scelta rapida predefiniti*.

Nella sezione *Personalizzazione barra multifunzione* si toglie il segno di spunta a tutte le caselle di controllo associate ai nomi delle schede principali.

Infine si chiude e si riapre il database, che dopo tutte queste modifiche dovrebbe presentarsi come nella Figura 6.26.

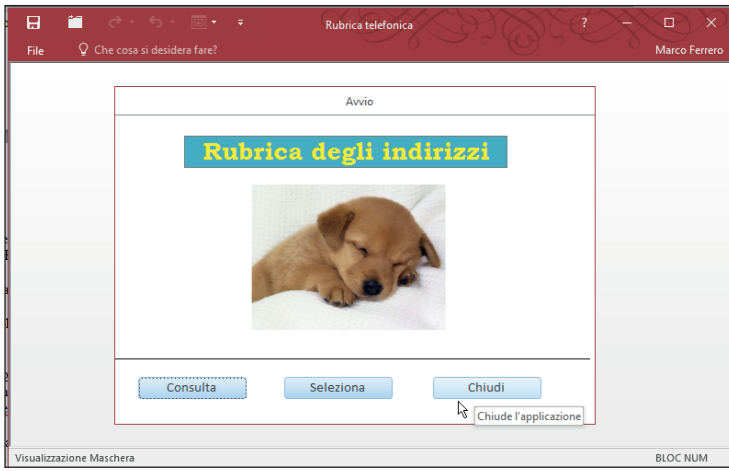


Figura 6.26 Un'applicazione aperta con impostazioni di Avvio personalizzate.

Dalla finestra dell'applicazione aperta in questo modo, l'operatore può agire soltanto sulla maschera Avvio (che viene aperta contestualmen-

te con l'apertura dell'applicazione) e tramite questa sulla maschera TBLINDIRIZZI, direttamente (premendo il pulsante *Consulta*) o con l'aiuto della maschera FRMSELETTORE (premendo il pulsante *Seleziona*)

Nella finestra Access non sono visualizzati il Riquadro di spostamento e la barra multifunzione è ridotta a una sola scheda. Se l'operatore preme il pulsante *Esci* l'applicazione si chiude e si chiude anche Access. Se apre la finestra *Backstage*, questa gli offre soltanto la possibilità di chiudere il database.

Le impostazioni predisposte nel modo che abbiamo descritto si attivano automaticamente quando si apre un'applicazione. Se fosse necessario intervenire su un'applicazione che è stata privata, come quella dell'esempio, di tutte le funzionalità di modifica, basta disattivare le impostazioni per l'avvio ricorrendo a un semplice trucco: dopo aver aperto Access, si seleziona l'applicazione che interessa nella finestra di dialogo *Apri* e in questa si dà il comando *Apri* tenendo premuto il tasto Maiusc; l'applicazione si apre nel contesto completo di Access, con tutta la sua strumentazione attiva. Sarà bene evitare che gli utenti finali apprendano questo trucco.