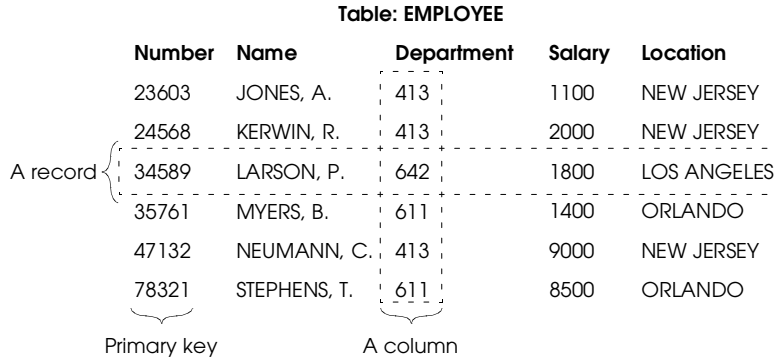

Fig. 18.1 Relational database structure.

Table: DEPARTMENT-LOCATOR

Department	Location
413	NEW JERSEY
611	ORLANDO
642	LOS ANGELES

Fig. 18.2 A table formed by projection.

Field	Description
`AuthorID`	The author's ID number in the database. This is the primary key field for this table.
`FirstName`	The author's first name.
`LastName`	The author's last name.
`YearBorn`	The author's year of birth.

Fig. 18.3 `Authors` table from `Books.mdb`.

AuthorID	FirstName	LastName	YearBorn
1	Harvey	Deitel	1946
2	Paul	Deitel	1968
3	Tem	Nieto	1969

Fig. 18.4 Data from the **Authors** table of **Books.mdb**.

Field	Description
`PublisherID`	The publisher's ID number in the database. This is the primary key field for this table.
`PublisherName`	The abbreviated name for the publisher.

Fig. 18.5 `Publishers` table from `Books.mdb`.

PublisherID	PublisherName
1	Prentice Hall
2	Prentice Hall PTR

Fig. 18.6 Data from the `Publishers` table of `Books.mdb`.

Field	Description
`ISBN`	The ISBN number for a book.
`AuthorID`	The author's ID number, which allows the database to connect each book to a specific author. The ID number in this field must also appear in the `Authors` table.

Fig. 18.7 `AuthorISBN` table from `Books.mdb`.

ISBN	AuthorID	ISBN	AuthorID
0-13-010671-2	1	0-13-020522-2	3
0-13-010671-2	2	0-13-082714-2	1
0-13-020522-2	1	0-13-082714-2	2
0-13-020522-2	2	0-13-082925-0	1
0-13-082925-0	2	0-13-565912-4	2
0-13-082927-7	1	0-13-565912-4	3
0-13-082927-7	2	0-13-899394-7	1
0-13-082928-5	1	0-13-899394-7	2
0-13-082928-5	2	0-13-904947-9	1
0-13-082928-5	3	0-13-904947-9	2
0-13-083054-2	1	0-13-904947-9	3
0-13-083054-2	2	0-13-GSVCPP-x	1
0-13-083055-0	1	0-13-GSVCPP-x	2
0-13-083055-0	2	0-13-IWCTC-x	1
0-13-118043-6	1	0-13-IWCTC-x	2
0-13-118043-6	2	0-13-IWCTC-x	3
0-13-226119-7	1	0-13-IWWW-x	1
0-13-226119-7	2	0-13-IWWW-x	2
0-13-271974-6	1	0-13-IWWW-x	3
0-13-271974-6	2	0-13-IWWWIM-x	1
0-13-456955-5	1	0-13-IWWWIM-x	2
0-13-456955-5	2	0-13-IWWWIM-x	3
0-13-456955-5	3	0-13-JAVA3-x	1
0-13-528910-6	1	0-13-JAVA3-x	2
0-13-528910-6	2	0-13-JCTC2-x	1
0-13-565912-4	1	0-13-JCTC2-x	2

Fig. 18.8 Data from the **AuthorISBN** table of **Books.mdb**.

Field	Description
`ISBN`	ISBN number of the book.
`Title`	Title of the book.
`EditionNumber`	The edition number of the book.
`YearPublished`	Year in which the book was published.
`Description`	A description of the book.
`PublisherID`	The publisher's ID number. This value must correspond to an ID number in the `Publishers` table.

Fig. 18.9 `Titles` table from `Books.mdb`.

ISBN	Title	Edition-Number	Year-Published	PublisherID
0-13-226119-7	C How to Program	2	1994	1
0-13-528910-6	C++ How to Program	2	1997	1
0-13-899394-7	Java How to Program	2	1997	1
0-13-java3-x	Java How to Program	3	1999	1
0-13-456955-5	Visual Basic 6 How to Program	1	1998	1
0-13-iwww-x	Internet and World Wide Web How to Program	1	1999	1
0-13-gsvcpp-x	Getting Started with Visual C++ 6 with an Introduction to MFC	1	1999	1
0-13-565912-4	C++ How to Program Instructor's Manual with Solutions Disk	2	1998	1
0-13-904947-9	Java How to Program Instructor's Manual with Solution Disk	2	1997	1
0-13-020522-2	Visual Basic 6 How to Program Instructor's Manual with Solution Disk	1	1999	1
0-13-iwwwim-x	Internet and World Wide Web How to Program Instructor's Manual with Solutions Disk	1	1999	1
0-13-082925-0	The Complete C++ Training Course	2	1998	2
0-13-082927-7	The Complete Java Training Course	2	1997	2
0-13-082928-5	The Complete Visual Basic 6 Training Course	1	1999	2
0-13-jctc2-x	The Complete Java Training Course	3	1999	2
0-13-iwctc-x	The Internet and World Wide Web How to Program Complete Training Course	1	1999	2
0-13-082714-2	C++ How to Program 2/e and Getting Started with Visual C++ 5.0 Tutorial	2	1998	1
0-13-010671-2	Java How to Program 2/e and Getting Started with Visual J++ 1.1 Tutorial	2	1998	1
0-13-083054-2	The Complete C++ Training Course 2/e and Getting Started with Visual C++ 5.0 Tutorial	2	1998	1
0-13-083055-0	The Complete Java Training Course 2/e and Getting Started with Visual J++ 1.1 Tutorial	2	1998	1
0-13-118043-6	C How to Program	1	1992	1
0-13-271974-6	Java Multimedia Cyber Classroom	1	1996	2

Fig. 18.10 Data from the `Titles` table of `Books.mdb`.

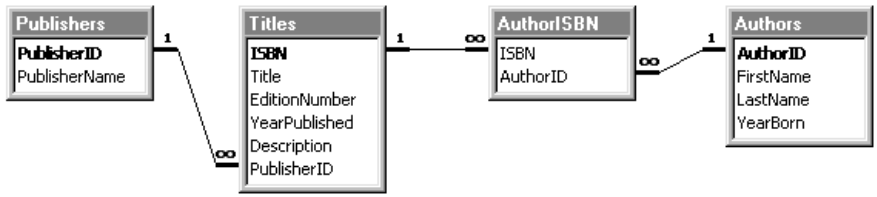

Fig. 18.11 Table relationships in **`Books.mdb`**.

SQL keyword	Description
`SELECT`	Select (retrieve) fields from one or more tables.
`FROM`	Tables from which to get fields. Required in every `SELECT`.
`WHERE`	Criteria for selection that determine the rows to be retrieved.
`GROUP BY`	How to group records.
`HAVING`	Used with the `GROUP BY` clause to specify criteria for grouping records in the query results.
`ORDER BY`	Criteria for ordering of records.

Fig. 18.12 SQL query keywords.

AuthorID	LastName
1	Deitel
2	Deitel
3	Nieto

Fig. 18.13 `AuthorID` and `LastName` from the `Authors` table.

AuthorID	FirstName	LastName	YearBorn
2	Paul	Deitel	1968
3	Tem	Nieto	1969

Fig. 18.14 Authors born after 1960 from the **Authors** table.

AuthorID	FirstName	LastName	YearBorn
1	Harvey	Deitel	1946
2	Paul	Deitel	1968

Fig. 18.15 Authors whose last names start with `d` from the `Authors` table.

AuthorID	FirstName	LastName	YearBorn
3	Tem	Nieto	1969

Fig. 18.16 Authors from the **Authors** table whose last names contain **i** as the second letter.

AuthorID	FirstName	LastName	YearBorn
2	Paul	Deitel	1968
1	Harvey	Deitel	1946
3	Tem	Nieto	1969

Fig. 18.17 Authors from the **Authors** table in ascending order by **LastName**.

AuthorID	FirstName	LastName	YearBorn
3	Tem	Nieto	1969
2	Paul	Deitel	1968
1	Harvey	Deitel	1946

Fig. 18.18 Authors from the **Authors** table in descending order by **LastName**.

AuthorID	FirstName	LastName	YearBorn
1	Harvey	Deitel	1946
2	Paul	Deitel	1968
3	Tem	Nieto	1969

Fig. 18.19 Authors from the **Authors** table in ascending order by **LastName** and by **FirstName**.

ISBN	Title	Edition-Number	Year-Published	PublisherID
0-13-118043-6	C How to Program	1	1992	1
0-13-226119-7	C How to Program	2	1994	1
0-13-528910-6	C++ How to Program	2	1997	1
0-13-iwww-x	Internet and World Wide Web How to Program	1	1999	1
0-13-java3-x	Java How to Program	3	1999	1
0-13-899394-7	Java How to Program	2	1997	1
0-13-456955-5	Visual Basic 6 How to Program	1	1998	1

Fig. 18.20 Books from the `Titles` table whose titles end with `How to Program` in ascending order by `Title`.

FirstName	LastName	ISBN	FirstName	LastName	ISBN
Harvey	Deitel	0-13-gsvcpp-x	Harvey	Deitel	0-13-010671-2
Harvey	Deitel	0-13-271974-6	Harvey	Deitel	0-13-118043-6
Harvey	Deitel	0-13-528910-6	Paul	Deitel	0-13-082928-5
Harvey	Deitel	0-13-083055-0	Paul	Deitel	0-13-082925-0
Harvey	Deitel	0-13-565912-4	Paul	Deitel	0-13-020522-2
Harvey	Deitel	0-13-083054-2	Paul	Deitel	0-13-904947-9
Harvey	Deitel	0-13-899394-7	Paul	Deitel	0-13-java3-x
Harvey	Deitel	0-13-904947-9	Paul	Deitel	0-13-iwwwim-x
Harvey	Deitel	0-13-226119-7	Paul	Deitel	0-13-iwww-x
Harvey	Deitel	0-13-082928-5	Paul	Deitel	0-13-iwctc-x
Harvey	Deitel	0-13-456955-5	Paul	Deitel	0-13-gsvcpp-x
Harvey	Deitel	0-13-iwwwim-x	Paul	Deitel	0-13-226119-7
Harvey	Deitel	0-13-iwctc-x	Paul	Deitel	0-13-899394-7
Harvey	Deitel	0-13-jctc2-x	Paul	Deitel	0-13-565912-4
Harvey	Deitel	0-13-082925-0	Paul	Deitel	0-13-528910-6
Harvey	Deitel	0-13-iwww-x	Paul	Deitel	0-13-jctc2-x
Harvey	Deitel	0-13-082714-2	Paul	Deitel	0-13-456955-5
Harvey	Deitel	0-13-082927-7	Paul	Deitel	0-13-271974-6
Harvey	Deitel	0-13-java3-x	Tem	Nieto	0-13-082928-5
Harvey	Deitel	0-13-020522-2	Tem	Nieto	0-13-565912-4
Paul	Deitel	0-13-118043-6	Tem	Nieto	0-13-456955-5
Paul	Deitel	0-13-010671-2	Tem	Nieto	0-13-iwctc-x
Paul	Deitel	0-13-083055-0	Tem	Nieto	0-13-iwww-x
Paul	Deitel	0-13-082927-7	Tem	Nieto	0-13-020522-2
Paul	Deitel	0-13-083054-2	Tem	Nieto	0-13-iwwwim-x
Paul	Deitel	0-13-082714-2	Tem	Nieto	0-13-904947-9

Fig. 18.21 Authors and the ISBN numbers for the books they have written in ascending order by **`LastName`** and **`FirstName`**.

```
SELECT Titles.Title, Titles.ISBN, Authors.FirstName,
       Authors.LastName, Titles.YearPublished,
       Publishers.PublisherName
FROM
   (Publishers INNER JOIN Titles
      ON Publishers.PublisherID = Titles.PublisherID)
   INNER JOIN
   (Authors INNER JOIN AuthorISBN ON
      Authors.AuthorID = AuthorISBN.AuthorID)
   ON Titles.ISBN = AuthorISBN.ISBN
ORDER BY Titles.Title
```

Fig. 18.22 The **`TitleAuthor`** query from the **`Books.mdb`** database.

Title	ISBN	First Name	Last-Name	Year-Published	Publisher-Name
C How to Program	0-13-226119-7	Paul	Deitel	1994	Prentice Hall
C How to Program	0-13-118043-6	Paul	Deitel	1992	Prentice Hall
C How to Program	0-13-118043-6	Harvey	Deitel	1992	Prentice Hall
C How to Program	0-13-226119-7	Harvey	Deitel	1994	Prentice Hall
C++ How to Program	0-13-528910-6	Harvey	Deitel	1997	Prentice Hall
C++ How to Program	0-13-528910-6	Paul	Deitel	1997	Prentice Hall
…					
Internet and World Wide Web How to Program	0-13-IWWW-x	Paul	Deitel	1999	Prentice Hall
Internet and World Wide Web How to Program	0-13-IWWW-x	Harvey	Deitel	1999	Prentice Hall
Internet and World Wide Web How to Program	0-13-IWWW-x	Tem	Nieto	1999	Prentice Hall
…					
Java How to Program	0-13-JAVA3-x	Harvey	Deitel	1999	Prentice Hall
Java How to Program	0-13-899394-7	Paul	Deitel	1997	Prentice Hall
Java How to Program	0-13-899394-7	Harvey	Deitel	1997	Prentice Hall
Java How to Program	0-13-JAVA3-x	Paul	Deitel	1999	Prentice Hall
…					
Visual Basic 6 How to Program	0-13-456955-5	Harvey	Deitel	1998	Prentice Hall
Visual Basic 6 How to Program	0-13-456955-5	Paul	Deitel	1998	Prentice Hall
Visual Basic 6 How to Program	0-13-456955-5	Tem	Nieto	1998	Prentice Hall

Fig. 18.23 A portion of the query results from the `TitleAuthor` query.

```
1  // Fig. 18.24: TableDisplay.java
2  // This program displays the contents of the Authors table
3  // in the Books database.
4  import java.sql.*;
5  import javax.swing.*;
6  import java.awt.*;
7  import java.awt.event.*;
8  import java.util.*;
9
10 public class TableDisplay extends JFrame {
11    private Connection connection;
12    private JTable table;
13
14    public TableDisplay()
15    {
16       // The URL specifying the Books database to which
17       // this program connects using JDBC to connect to a
18       // Microsoft ODBC database.
19       String url = "jdbc:odbc:Books";
20       String username = "anonymous";
21       String password = "guest";
22
23       // Load the driver to allow connection to the database
24       try {
25          Class.forName( "sun.jdbc.odbc.JdbcOdbcDriver" );
26
27          connection = DriverManager.getConnection(
28             url, username, password );
29       }
30       catch ( ClassNotFoundException cnfex ) {
31          System.err.println(
32             "Failed to load JDBC/ODBC driver." );
33          cnfex.printStackTrace();
34          System.exit( 1 );  // terminate program
35       }
36       catch ( SQLException sqlex ) {
37          System.err.println( "Unable to connect" );
38          sqlex.printStackTrace();
39       }
40
```

Fig. 18.24 Connecting to a database, querying the database and displaying the results (part 1 of 4).

```
41       getTable();
42
43       setSize( 450, 150 );
44       show();
45    }
46
47    private void getTable()
48    {
49       Statement statement;
```

```
50          ResultSet resultSet;
51
52          try {
53             String query = "SELECT * FROM Authors";
54
55             statement = connection.createStatement();
56             resultSet = statement.executeQuery( query );
57             displayResultSet( resultSet );
58             statement.close();
59          }
60          catch ( SQLException sqlex ) {
61             sqlex.printStackTrace();
62          }
63       }
64
65       private void displayResultSet( ResultSet rs )
66          throws SQLException
67       {
68          // position to first record
69          boolean moreRecords = rs.next();
70
71          // If there are no records, display a message
72          if ( ! moreRecords ) {
73             JOptionPane.showMessageDialog( this,
74                "ResultSet contained no records" );
75             setTitle( "No records to display" );
76             return;
77          }
78
79          setTitle( "Authors table from Books" );
80
81          Vector columnHeads = new Vector();
82          Vector rows = new Vector();
83
84          try {
85             // get column heads
86             ResultSetMetaData rsmd = rs.getMetaData();
87
88             for ( int i = 1; i <= rsmd.getColumnCount(); ++i )
89                columnHeads.addElement( rsmd.getColumnName( i ) );
90
```

Fig. 18.24 Connecting to a database, querying the database and displaying the results (part 2 of 4).

```
91             // get row data
92             do {
93                rows.addElement( getNextRow( rs, rsmd ) );
94             } while ( rs.next() );
95
96             // display table with ResultSet contents
97             table = new JTable( rows, columnHeads );
98             JScrollPane scroller = new JScrollPane( table );
99             getContentPane().add(
```

```
100             scroller, BorderLayout.CENTER );
101          validate();
102       }
103       catch ( SQLException sqlex ) {
104          sqlex.printStackTrace();
105       }
106    }
107
108    private Vector getNextRow( ResultSet rs,
109                               ResultSetMetaData rsmd )
110        throws SQLException
111    {
112       Vector currentRow = new Vector();
113
114       for ( int i = 1; i <= rsmd.getColumnCount(); ++i )
115          switch( rsmd.getColumnType( i ) ) {
116             case Types.VARCHAR:
117                   currentRow.addElement( rs.getString( i ) );
118                break;
119             case Types.INTEGER:
120                   currentRow.addElement(
121                      new Long( rs.getLong( i ) ) );
122                break;
123             default:
124                System.out.println( "Type was: " +
125                   rsmd.getColumnTypeName( i ) );
126          }
127
128       return currentRow;
129    }
130
131    public void shutDown()
132    {
133       try {
134          connection.close();
135       }
136       catch ( SQLException sqlex ) {
137          System.err.println( "Unable to disconnect" );
138          sqlex.printStackTrace();
139       }
140    }
141
```

Fig. 18.24 Connecting to a database, querying the database and displaying the results (part 3 of 4).

```
142    public static void main( String args[] )
143    {
144       final TableDisplay app = new TableDisplay();
145
146       app.addWindowListener(
147          new WindowAdapter() {
148             public void windowClosing( WindowEvent e )
149             {
150                app.shutDown();
```

```
151                     System.exit( 0 );
152                  }
153               }
154            );
155      }
156   }
```

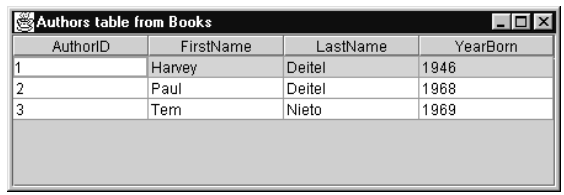

Fig. 18.24 Connecting to a database, querying the database and displaying the results (part 4 of 4).

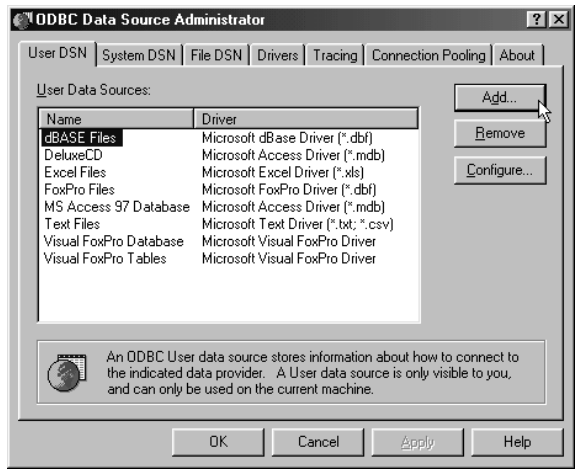

Fig. 18.25 ODBC Data Source Administrator dialog.

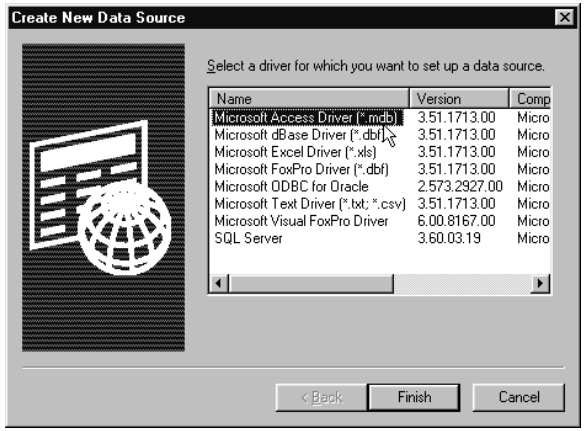

Fig. 18.26 Create New Data Source dialog.

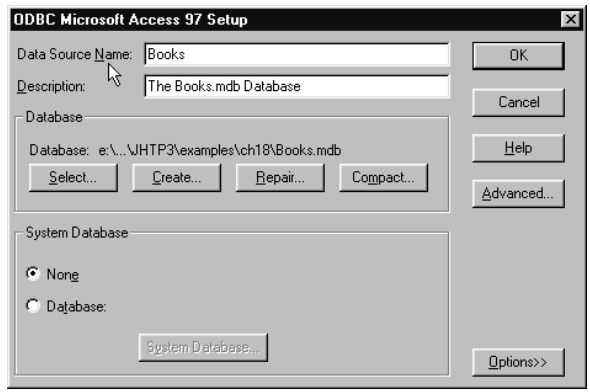

Fig. 18.27 ODBC Microsoft Access 97 Setup dialog.

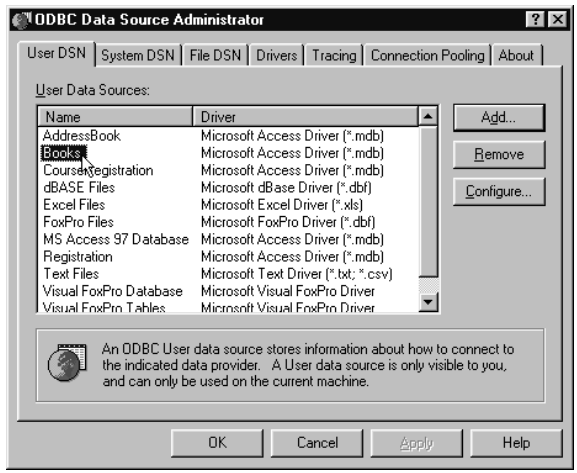

Fig. 18.28 ODBC Data Source Administrator dialog displaying registered drivers.

```
1  // Fig. 18.29: DisplayQueryResults.java
2  // This program displays the ResultSet returned by a
3  // query on the Books database.
4  import java.sql.*;
5  import javax.swing.*;
6  import java.awt.*;
7  import java.awt.event.*;
8  import java.util.*;
9
10 public class DisplayQueryResults extends JFrame {
11    // java.sql types needed for database processing
12    private Connection connection;
13    private Statement statement;
14    private ResultSet resultSet;
```

Fig. 18.29 Submitting queries to the **Books.mdb** database (part 1 of 5).

```
15    private ResultSetMetaData rsMetaData;
16
17    // javax.swing types needed for GUI
18    private JTable table;
19    private JTextArea inputQuery;
20    private JButton submitQuery;
21
22    public DisplayQueryResults()
23    {
24       super( "Enter Query. Click Submit to See Results." );
25
26       // The URL specifying the Books database to which
27       // this program connects using JDBC to connect to a
28       // Microsoft ODBC database.
29       String url = "jdbc:odbc:Books";
30       String username = "anonymous";
31       String password = "guest";
32
33       // Load the driver to allow connection to the database
34       try {
35          Class.forName( "sun.jdbc.odbc.JdbcOdbcDriver" );
36
37          connection = DriverManager.getConnection(
38             url, username, password );
39       }
40       catch ( ClassNotFoundException cnfex ) {
41          System.err.println(
42             "Failed to load JDBC/ODBC driver." );
43          cnfex.printStackTrace();
44          System.exit( 1 );  // terminate program
45       }
46       catch ( SQLException sqlex ) {
47          System.err.println( "Unable to connect" );
48          sqlex.printStackTrace();
49          System.exit( 1 );  // terminate program
50       }
51
52       // If connected to database, set up GUI
```

```
53          inputQuery =
54             new JTextArea( "SELECT * FROM Authors", 4, 30 );
55          submitQuery = new JButton( "Submit query" );
56          submitQuery.addActionListener(
57             new ActionListener() {
58                public void actionPerformed( ActionEvent e )
59                {
60                   if ( e.getSource() == submitQuery )
61                      getTable();
62                }
63             }
64          );
65
66          JPanel topPanel = new JPanel();
67          topPanel.setLayout( new BorderLayout() );
```

Fig. 18.29 Submitting queries to the **Books.mdb** database (part 2 of 5).

```
68          topPanel.add( new JScrollPane( inputQuery),
69                        BorderLayout.CENTER );
70          topPanel.add( submitQuery, BorderLayout.SOUTH );
71
72          table = new JTable( 4, 4 );
73
74          Container c = getContentPane();
75          c.setLayout( new BorderLayout() );
76          c.add( topPanel, BorderLayout.NORTH );
77          c.add( table, BorderLayout.CENTER );
78
79          getTable();
80
81          setSize( 500, 500 );
82          show();
83       }
84
85       private void getTable()
86       {
87          try {
88             String query = inputQuery.getText();
89
90             statement = connection.createStatement();
91             resultSet = statement.executeQuery( query );
92             displayResultSet( resultSet );
93          }
94          catch ( SQLException sqlex ) {
95             sqlex.printStackTrace();
96          }
97       }
98
99       private void displayResultSet( ResultSet rs )
100         throws SQLException
101      {
102         // position to first record
103         boolean moreRecords = rs.next();
104
```

```
105         // If there are no records, display a message
106         if ( ! moreRecords ) {
107            JOptionPane.showMessageDialog( this,
108               "ResultSet contained no records" );
109            setTitle( "No records to display" );
110            return;
111         }
112
113         Vector columnHeads = new Vector();
114         Vector rows = new Vector();
115
116         try {
117            // get column heads
118            ResultSetMetaData rsmd = rs.getMetaData();
119
```

Fig. 18.29 Submitting queries to the **Books.mdb** database (part 3 of 5).

```
120            for ( int i = 1; i <= rsmd.getColumnCount(); ++i )
121               columnHeads.addElement( rsmd.getColumnName( i ) );
122
123            // get row data
124            do {
125               rows.addElement( getNextRow( rs, rsmd ) );
126            } while ( rs.next() );
127
128            // display table with ResultSet contents
129            table = new JTable( rows, columnHeads );
130            JScrollPane scroller = new JScrollPane( table );
131            Container c = getContentPane();
132            c.remove( 1 );
133            c.add( scroller, BorderLayout.CENTER );
134            c.validate();
135         }
136         catch ( SQLException sqlex ) {
137            sqlex.printStackTrace();
138         }
139      }
140
141      private Vector getNextRow( ResultSet rs,
142                                 ResultSetMetaData rsmd )
143          throws SQLException
144      {
145         Vector currentRow = new Vector();
146
147         for ( int i = 1; i <= rsmd.getColumnCount(); ++i )
148            switch( rsmd.getColumnType( i ) ) {
149               case Types.VARCHAR:
150               case Types.LONGVARCHAR:
151                     currentRow.addElement( rs.getString( i ) );
152                  break;
153               case Types.INTEGER:
154                     currentRow.addElement(
155                        new Long( rs.getLong( i ) ) );
```

```
156                break;
157             default:
158                System.out.println( "Type was: " +
159                   rsmd.getColumnTypeName( i ) );
160          }
161
162       return currentRow;
163    }
164
165    public void shutDown()
166    {
167       try {
168          connection.close();
169       }
170       catch ( SQLException sqlex ) {
171          System.err.println( "Unable to disconnect" );
```

Fig. 18.29 Submitting queries to the **`Books.mdb`** database (part 4 of 5).

```
172          sqlex.printStackTrace();
173       }
174    }
175
176    public static void main( String args[] )
177    {
178       final DisplayQueryResults app =
179          new DisplayQueryResults();
180
181       app.addWindowListener(
182          new WindowAdapter() {
183             public void windowClosing( WindowEvent e )
184             {
185                app.shutDown();
186                System.exit( 0 );
187             }
188          }
189       );
190    }
191 }
```

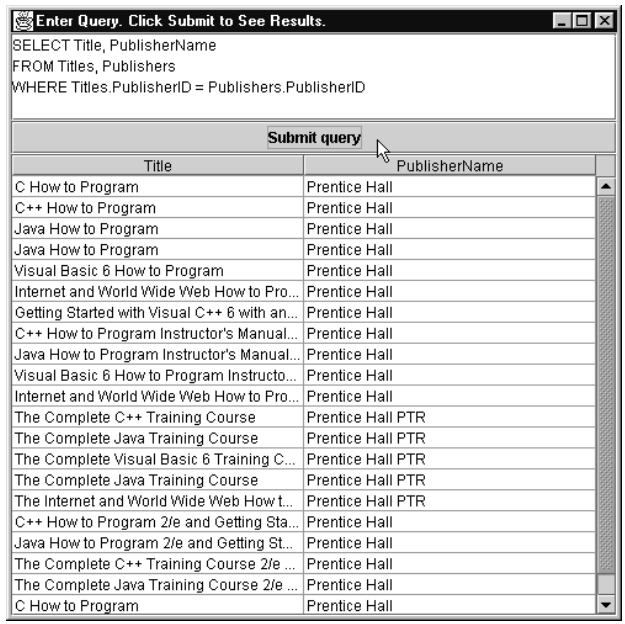

Fig. 18.29 Submitting queries to the **`Books.mdb`** database (part 5 of 5).

```
1  // Fig. 18.30: AddressBook.java
2  // Inserting into, updating and searching through a database
3  import java.sql.*;
4  import java.awt.*;
5  import java.awt.event.*;
6  import javax.swing.*;
7
8  public class AddressBook extends JFrame {
9     private ControlPanel controls;
10    private ScrollingPanel scrollArea;
11    private JTextArea output;
12    private String url;
13    private Connection connect;
14    private JScrollPane textpane;
15
16    public AddressBook()
17    {
18       super( "Address Book Database Application" );
19
20       Container c = getContentPane();
21
22       // Start screen layout
23       scrollArea = new ScrollingPanel();
24       output = new JTextArea( 6, 30 );
25       c.setLayout( new BorderLayout() );
26       c.add( new JScrollPane( scrollArea ),
27              BorderLayout.CENTER );
28       textpane = new JScrollPane( output );
29       c.add( textpane, BorderLayout.SOUTH );
30
31       // Set up database connection
32       try {
33          url = "jdbc:odbc:AddressBook";
34
35          Class.forName( "sun.jdbc.odbc.JdbcOdbcDriver" );
36          connect = DriverManager.getConnection( url );
37          output.append( "Connection successful\n" );
38       }
39       catch ( ClassNotFoundException cnfex ) {
40          // process ClassNotFoundExceptions here
41          cnfex.printStackTrace();
42          output.append( "Connection unsuccessful\n" +
43                         cnfex.toString() );
44       }
```

Fig. 18.30 Inserting, finding and updating records (part 1 of 16).

```
45       catch ( SQLException sqlex ) {
46          // process SQLExceptions here
47          sqlex.printStackTrace();
48          output.append( "Connection unsuccessful\n" +
49                         sqlex.toString() );
50       }
```

```
51          catch ( Exception ex ) {
52             // process remaining Exceptions here
53             ex.printStackTrace();
54             output.append( ex.toString() );
55          }
56
57          // Complete screen layout
58          controls =
59             new ControlPanel( connect, scrollArea, output);
60          c.add( controls, BorderLayout.NORTH );
61
62          setSize( 500, 500 );
63          show();
64       }
65
66       public static void main( String args[] )
67       {
68          AddressBook app = new AddressBook();
69
70          app.addWindowListener(
71             new WindowAdapter() {
72                public void windowClosing( WindowEvent e )
73                {
74                   System.exit( 0 );
75                }
76             }
77          );
78       }
79    }
```

Fig. 18.30 Inserting, finding and updating records (part 2 of 16).

```
80    // Fig. 18.30: AddRecord.java
81    // Class AddRecord definition
82    import java.awt.*;
83    import java.awt.event.*;
84    import java.sql.*;
85    import javax.swing.*;
86
87    public class AddRecord implements ActionListener {
88       private ScrollingPanel fields;
89       private JTextArea output;
90       private Connection connection;
91
```

Fig. 18.30 Inserting, finding and updating records (part 3 of 16).

```
92       public AddRecord( Connection c, ScrollingPanel f,
93                         JTextArea o )
94       {
95          connection = c;
96          fields = f;
97          output = o;
98       }
```

```
99
100     public void actionPerformed( ActionEvent e )
101     {
102        try {
103           Statement statement = connection.createStatement();
104
105           if ( !fields.last.getText().equals( "" ) &&
106                !fields.first.getText().equals( "" ) ) {
107              String query = "INSERT INTO addresses (" +
108                 "firstname, lastname, address, city, " +
109                 "stateorprovince, postalcode, country, " +
110                 "emailaddress, homephone, faxnumber" +
111                 ") VALUES ('" +
112                 fields.first.getText() + "', '" +
113                 fields.last.getText() + "', '" +
114                 fields.address.getText() + "', '" +
115                 fields.city.getText() + "', '" +
116                 fields.state.getText() + "', '" +
117                 fields.zip.getText() + "', '" +
118                 fields.country.getText() + "', '" +
119                 fields.email.getText() + "', '" +
120                 fields.home.getText() + "', '" +
121                 fields.fax.getText() + "')";
122              output.append( "\nSending query: " +
123                             connection.nativeSQL( query )
124                             + "\n" );
125              int result = statement.executeUpdate( query );
126
127              if ( result == 1 )
128                 output.append( "\nInsertion successful\n" );
129              else {
130                 output.append( "\nInsertion failed\n" );
131                 fields.first.setText( "" );
132                 fields.last.setText( "" );
133                 fields.address.setText( "" );
134                 fields.city.setText( "" );
135                 fields.state.setText( "" );
136                 fields.zip.setText( "" );
137                 fields.country.setText( "" );
138                 fields.email.setText( "" );
139                 fields.home.setText( "" );
140                 fields.fax.setText( "" );
141              }
142           }
```

Fig. 18.30 Inserting, finding and updating records (part 4 of 16).

```
143           else
144              output.append( "\nEnter at least first and " +
145                             "last name then press Add\n" );
146
147           statement.close();
148        }
149        catch ( SQLException sqlex ) {
150           sqlex.printStackTrace();
```

```
            output.append( sqlex.toString() );
         }
      }
   }
```

Fig. 18.30 Inserting, finding and updating records (part 5 of 16).

```
// Fig. 18.30: FindRecord.java
// Class FindRecord defintion
import java.awt.*;
import java.awt.event.*;
import java.sql.*;
import javax.swing.*;

public class FindRecord implements ActionListener {
   private ScrollingPanel fields;
   private JTextArea output;
   private Connection connection;

   public FindRecord( Connection c, ScrollingPanel f,
                      JTextArea o )
   {
      connection = c;
      fields = f;
      output = o;
   }

   public void actionPerformed( ActionEvent e )
   {
      try {
         if ( !fields.last.getText().equals( "" ) ) {
            Statement statement =connection.createStatement();
            String query = "SELECT * FROM addresses " +
                           "WHERE lastname = '" +
                           fields.last.getText() + "'";
            output.append( "\nSending query: " +
                           connection.nativeSQL( query )
                           + "\n" );
            ResultSet rs = statement.executeQuery( query );
            display( rs );
            output.append( "\nQuery successful\n" );
            statement.close();
         }
```

Fig. 18.30 Inserting, finding and updating records (part 6 of 16).

```
         else
            fields.last.setText(
               "Enter last name here then press Find" );
      }
      catch ( SQLException sqlex ) {
         sqlex.printStackTrace();
         output.append( sqlex.toString() );
      }
```

```
199     }
200
201     // Display results of query. If rs is null
202     public void display( ResultSet rs )
203     {
204        try {
205           rs.next();
206
207           int recordNumber = rs.getInt( 1 );
208
209           if ( recordNumber != 0 ) {
210              fields.id.setText( String.valueOf( recordNumber));
211              fields.first.setText( rs.getString( 2 ) );
212              fields.last.setText( rs.getString( 3 ) );
213              fields.address.setText( rs.getString( 4 ) );
214              fields.city.setText( rs.getString( 5 ) );
215              fields.state.setText( rs.getString( 6 ) );
216              fields.zip.setText( rs.getString( 7 ) );
217              fields.country.setText( rs.getString( 8 ) );
218              fields.email.setText( rs.getString( 9 ) );
219              fields.home.setText( rs.getString( 10 ) );
220              fields.fax.setText( rs.getString( 11 ) );
221           }
222           else
223              output.append( "\nNo record found\n" );
224        }
225        catch ( SQLException sqlex ) {
226           sqlex.printStackTrace();
227           output.append( sqlex.toString() );
228        }
229     }
230  }
```

Fig. 18.30 Inserting, finding and updating records (part 7 of 16).

```
231  // Fig. 18.30: UpdateRecord.java
232  // Class UpdateRecord definition
233  import java.awt.*;
234  import java.awt.event.*;
235  import java.sql.*;
236  import javax.swing.*;
237
238  public class UpdateRecord implements ActionListener {
239     private ScrollingPanel fields;
```

Fig. 18.30 Inserting, finding and updating records (part 8 of 16).

```
240     private JTextArea output;
241     private Connection connection;
242
243     public UpdateRecord( Connection c, ScrollingPanel f,
244                          JTextArea o )
245     {
246        connection = c;
```

```
247         fields = f;
248         output = o;
249      }
250
251      public void actionPerformed( ActionEvent e )
252      {
253         try {
254            Statement statement = connection.createStatement();
255
256            if ( ! fields.id.getText().equals( "" ) ) {
257               String query = "UPDATE addresses SET " +
258                      "firstname='" + fields.first.getText() +
259                      "', lastname='" + fields.last.getText() +
260                      "', address='" + fields.address.getText() +
261                      "', city='" + fields.city.getText() +
262                      "', stateorprovince='" +
263                      fields.state.getText() +
264                      "', postalcode='" + fields.zip.getText() +
265                      "', country='" + fields.country.getText() +
266                      "', emailaddress='" +
267                      fields.email.getText() +
268                      "', homephone='" + fields.home.getText() +
269                      "', faxnumber='" + fields.fax.getText() +
270                      "' WHERE id=" + fields.id.getText();
271                     output.append( "\nSending query: " +
272                     connection.nativeSQL( query ) + "\n" );
273
274               int result = statement.executeUpdate( query );
275
276               if ( result == 1 )
277                  output.append( "\nUpdate successful\n" );
278               else {
279                  output.append( "\nUpdate failed\n" );
280                  fields.first.setText( "" );
281                  fields.last.setText( "" );
282                  fields.address.setText( "" );
283                  fields.city.setText( "" );
284                  fields.state.setText( "" );
285                  fields.zip.setText( "" );
286                  fields.country.setText( "" );
287                  fields.email.setText( "" );
288                  fields.home.setText( "" );
289                  fields.fax.setText( "" );
290               }
291
```

Fig. 18.30 Inserting, finding and updating records (part 9 of 16).

```
292               statement.close();
293            }
294            else
295               output.append( "\nYou may only update an " +
296                              "existing record. Use Find to " +
297                              "locate the record, then " +
298                              "modify the information and " +
```

```
                              "press Update.\n" );
         }
         catch ( SQLException sqlex ) {
            sqlex.printStackTrace();
            output.append( sqlex.toString() );
         }
      }
   }
```

Fig. 18.30 Inserting, finding and updating records (part 10 of 16).

```
// Fig. 18.30: Help.java
// Class Help definition
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class Help implements ActionListener {
   private JTextArea output;

   public Help( JTextArea o )
   {
      output = o;
   }

   public void actionPerformed( ActionEvent e )
   {
      output.append( "\nClick Find to locate a record.\n" +
                     "Click Add to insert a new record.\n" +
                     "Click Update to update " +
                     "the information in a record.\n" +
                     "Click Clear to empty" +
                     " the textfields.\n" );
   }
}
```

Fig. 18.30 Inserting, finding and updating records (part 11 of 16).

```
// Fig. 18.30: ControlPanel.java
// Class ControlPanel definition
import java.awt.*;
import java.awt.event.*;
import java.sql.*;
import javax.swing.*;

```

Fig. 18.30 Inserting, finding and updating records (part 12 of 16).

```
public class ControlPanel extends JPanel {
   private JButton findName, addName,
                   updateName, clear, help;

   public ControlPanel( Connection c, ScrollingPanel s,
                        JTextArea t )
```

```
344    {
345       setLayout( new GridLayout( 1, 5 ) );
346
347       findName = new JButton( "Find" );
348       findName.addActionListener( new FindRecord( c, s, t ) );
349       add( findName );
350
351       addName = new JButton( "Add" );
352       addName.addActionListener( new AddRecord( c, s, t ) );
353       add( addName );
354
355       updateName = new JButton( "Update" );
356       updateName.addActionListener(
357          new UpdateRecord( c, s, t ) );
358       add( updateName );
359
360       clear = new JButton( "Clear" );
361       clear.addActionListener( new ClearFields( s ) );
362       add( clear );
363
364       help = new JButton( "Help" );
365       help.addActionListener( new Help( t ) );
366       add( help );
367    }
368 }
```

Fig. 18.30 Inserting, finding and updating records (part 13 of 16).

```
369 // Fig. 18.30: ScrollingPanel.java
370 // Class ScrollingPanel
371 import java.awt.*;
372 import java.awt.event.*;
373 import javax.swing.*;
374
375 public class ScrollingPanel extends JPanel {
376    private JPanel labelPanel, fieldsPanel;
377    private String labels[] =
378                  { "ID number:", "First name:", "Last name:",
379                    "Address:", "City:", "State/Province:",
380                    "PostalCode:", "Country:", "Email:",
381                    "Home phone:", "Fax Number:" };
382    JTextField id, first, last, address,     // package access
383               city, state, zip,
384               country, email, home, fax;
385
```

Fig. 18.30 Inserting, finding and updating records (part 14 of 16).

```
386    public ScrollingPanel()
387    {
388       // Label panel
389       labelPanel = new JPanel();
390       labelPanel.setLayout(
391                         new GridLayout( labels.length, 1 ) );
```

```
392
393         ImageIcon ii = new ImageIcon( "images/icon.jpg" );
394
395         for ( int i = 0; i < labels.length; i++ )
396            labelPanel.add( new JLabel( labels[ i ], ii, 0) );
397
398         // TextField panel
399         fieldsPanel = new JPanel();
400         fieldsPanel.setLayout(
401                           new GridLayout( labels.length, 1 ) );
402         id = new JTextField( 20 );
403         id.setEditable( false );
404         fieldsPanel.add( id );
405         first = new JTextField( 20 );
406         fieldsPanel.add( first );
407         last = new JTextField( 20 );
408         fieldsPanel.add( last );
409         address = new JTextField( 20 );
410         fieldsPanel.add( address );
411         city = new JTextField( 20 );
412         fieldsPanel.add( city );
413         state = new JTextField( 20 );
414         fieldsPanel.add( state  );
415         zip = new JTextField( 20 );
416         fieldsPanel.add( zip );
417         country = new JTextField( 20 );
418         fieldsPanel.add( country );
419         email = new JTextField( 20 );
420         fieldsPanel.add( email );
421         home = new JTextField( 20 );
422         fieldsPanel.add( home );
423         fax = new JTextField( 20 );
424         fieldsPanel.add( fax );
425
426         setLayout( new GridLayout( 1, 2 ) );
427         add( labelPanel );
428         add( fieldsPanel );
429      }
430   }
```

Fig. 18.30 Inserting, finding and updating records (part 15 of 16).

```
431   // Fig. 18.30: ClearFields.java
432   // Class ClearFields definition
433   import java.awt.*;
434   import java.awt.event.*;
435
436   public class ClearFields implements ActionListener {
437      private ScrollingPanel fields;
438
439      public ClearFields( ScrollingPanel f )
440      {
441         fields = f;
442      }
443
```

```
444     public void actionPerformed( ActionEvent e )
445     {
446        fields.id.setText( "" );
447        fields.first.setText( "" );
448        fields.last.setText( "" );
449        fields.address.setText( "" );
450        fields.city.setText( "" );
451        fields.state.setText( "" );
452        fields.zip.setText( "" );
453        fields.country.setText( "" );
454        fields.email.setText( "" );
455        fields.home.setText( "" );
456        fields.fax.setText( "" );
457     }
458  }
```

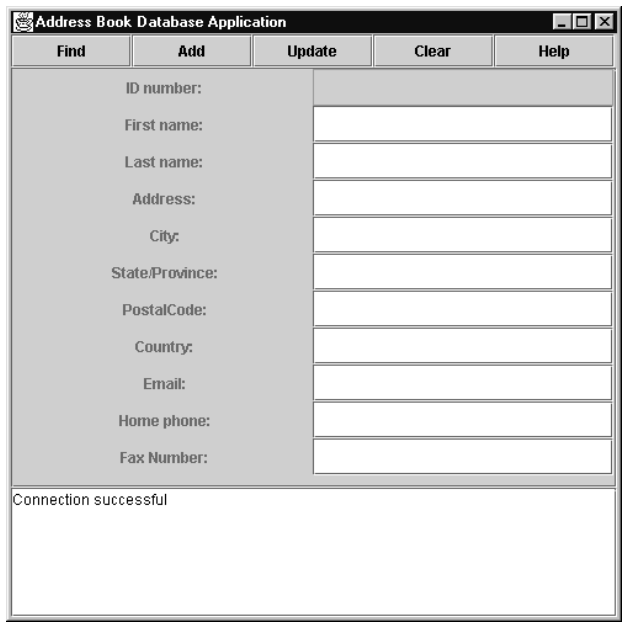

Fig. 18.30 Inserting, finding and updating records (part 16 of 16).