This paper analyzes Java Database Connectivity (JDBC), an object-oriented approach to connecting to databases in Java. JDBC is an object-oriented wrapping and redesign of ODBC API that is easier to learn and to use and that allows one to write vendor-independent code for querying and manipulating databases. JDBC provides some fairly low-level methods for accessing databases, as well as more powerful objects for handling databases at a higher level of abstraction.

What is a Database?

A database is a series of tables of information in some sort of file structure that allows you to access these tables, select columns from them, sort them, and select rows based on various criteria. Databases usually have indexes associated with many of the columns in these tables, so that we can access them as rapidly as possible.

Databases are used more than any other kind of structure in computing. You'll find databases as central elements of employee records and payroll systems, in travel-scheduling systems, and all through product manufacturing and marketing.

In the case of employee records, you could imagine a table of employee names and addresses and of salaries, tax withholding, and benefits. Let's consider how these might be organized. You can imagine one table of employee names, addresses and phone numbers. Other information that you might want to store would include salary, salary range, last raise, next raise, employee performance ranking, and so forth.

Should this all be in one table? Almost certainly not. Salary ranges for various employee types are probably invariant between employees; thus you would store only the employee type in the employee table and the salary ranges in another table, which is pointed to by the type number. Consider the following:

<table>
<thead>
<tr>
<th>Key</th>
<th>Lastname</th>
<th>SalaryType</th>
<th>SalaryType</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Adams</td>
<td>2</td>
<td>1</td>
<td>30000</td>
<td>45000</td>
</tr>
<tr>
<td>2</td>
<td>Johnson</td>
<td>1</td>
<td>2</td>
<td>45000</td>
<td>60000</td>
</tr>
<tr>
<td>3</td>
<td>Smyth</td>
<td>3</td>
<td>3</td>
<td>60000</td>
<td>75000</td>
</tr>
<tr>
<td>4</td>
<td>Tully</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Wolff</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The data in the SalaryType column refers to the second table. We could imagine many categories for such tables for things such as state of residence and tax values for each state, health plan withholding, and so forth. Each table will have a primary key column like the ones at the left of each table above and several more columns of data. Building tables in databases has evolved to both an art and a science. The structure of these tables is referred to by their normal form. Tables are said to be in first, second, or third normal form, abbreviated as 1NF, 2NF or 3NF.

1st. Each cell in a table should have only one value (never an array of values). (1NF)

2nd. 1NF and every non-key column is fully dependent on the key column. This means there is a 1-to-1 relationship between the primary key and the remaining cells in that row. (2NF)

3rd. 2NF and all non-key columns are mutually independent. This means that there are no data columns
containing values that can be calculated from other columns' data. (3NF)

Today nearly all databases are constructed so that all tables are in Third Normal Form (3NF). This means that there are usually a fairly large number of tables, each with relatively few columns of information.

### Getting Data out of Databases

Suppose we wanted to produce a table of employees and their salary ranges for some planning exercise. This table doesn't exist directly in the database, but it can be constructed by issuing a query to the database. We'd like to have a table that looked like the following:

<table>
<thead>
<tr>
<th>Name</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adams</td>
<td>$45,000.00</td>
<td>$60,000.00</td>
</tr>
<tr>
<td>Johnson</td>
<td>$30,000.00</td>
<td>$45,000.00</td>
</tr>
<tr>
<td>Smyth</td>
<td>$60,000.00</td>
<td>$75,000.00</td>
</tr>
<tr>
<td>Tully</td>
<td>$30,000.00</td>
<td>$45,000.00</td>
</tr>
<tr>
<td>Wolff</td>
<td>$45,000.00</td>
<td>$60,000.00</td>
</tr>
</tbody>
</table>

or maybe sorted by increasing salary as

<table>
<thead>
<tr>
<th>Name</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tully</td>
<td>$30,000.00</td>
<td>$45,000.00</td>
</tr>
<tr>
<td>Johnson</td>
<td>$30,000.00</td>
<td>$45,000.00</td>
</tr>
<tr>
<td>Wolff</td>
<td>$45,000.00</td>
<td>$60,000.00</td>
</tr>
<tr>
<td>Adams</td>
<td>$45,000.00</td>
<td>$60,000.00</td>
</tr>
<tr>
<td>Smyth</td>
<td>$60,000.00</td>
<td>$75,000.00</td>
</tr>
</tbody>
</table>

```sql
```

### Kinds of Databases

Since the PC became a major office tool, there have been a number of popular databases developed that are intended to run by themselves on PCs. These include elementary databases like Microsoft Works, as well as more sophisticated ones like Approach, dBase, Borland Paradox, Microsoft Access, and FoxBase.

Another category of PC databases includes those databases intended to be accessed from a server by a number of PC clients. These include IBM DB/2, Microsoft SQL Server, Oracle, Sybase, SQLBase, and XDB. All of these database products support various relatively similar dialects of SQL, and all of them thus would appear at first to be relatively interchangeable. The reason they are not interchangeable, of course, is that each was designed with different performance characteristics involved and each with a different user interface and programming interface. While you might think that since they all support SQL, programming them would be similar, quite the opposite is true, since each database has its own way of receiving the SQL queries and its own way of returning the results. This is where the next proposed level of standardization came about: ODBC.
ODBC

It would be nice if we could somehow write code that was independent of the particular vendor's database and that would allow us to get the same results from any of these databases without changing our calling program. If we could only write some wrappers for all of these databases so that they all appeared to have similar programming interfaces, this would be quite easy to accomplish.

Microsoft first attempted this feat in 1992, when they released a specification called Object Database Connectivity. It was supposed to be the answer for connection to all databases under Windows. Like all first software versions, this suffered some growing pains, and another version was released in 1994, which was somewhat faster, as well as more stable. It also was the first 32-bit version. In addition, ODBC began to move to other platforms than Windows and has by now become quite pervasive in the PC and Workstation world. ODBC drivers are provided by nearly every major database vendor.

However, ODBC is not the panacea we might at first suppose. Many database vendors support ODBC as an "alternate interface" to their standard one, and programming in ODBC is not trivial. It is much like other Windows programming, consisting of handles, pointers and options that make it hard to learn. Finally, ODBC is not an independently-controlled standard. It was developed and is being evolved by Microsoft, which, given the highly competitive software environment we all work in, makes its future hard to predict.

What Is JDBC?

JDBC is a set of initials that once stood for "Java DataBase Connectivity," but it is now a trademarked symbol on its own. It is an object-oriented wrapping and redesign of ODBC API that is much easier to learn and to use and that really does allow you both to write vendor-independent code to query and to manipulate databases. While it is object-oriented, as all Java APIs must be, it is not a very high level set of objects, and we will be developing some higher-level approaches in the course of the remainder of this chapter.

Most database vendors other than Microsoft have embraced JDBC and provide JDBC drivers for their databases; this makes it quite easy for you to really write almost completely database-independent code. In addition, JavaSoft and Intersolv have developed a product called the JDBC-ODBC Bridge, which allows you to connect to databases for which no direct JDBC driver yet exists. All of the databases which support JDBC must at a minimum support the SQL-92 standard. This makes for a great degree of portability across databases and platforms.

Installing and Using JDBC

The JDBC classes are grouped in the java.sql package and are installed when you install the Java JDK 1.1 or later. However, if you want to use the JDBC-ODBC bridge, you must install two other packages. First, if you are using Windows 95, you must upgrade your ODBC driver to the 32-bit driver, which you can download free from Microsoft's Web site. This driver is difficult to find on their site; search for DataAcc.exe and download and install it.

The JDBC-ODBC Driver is available from the Sun Java site (http://java.sun.com) and is easily located and downloaded. After you expand and install this driver, you must do the following:

1. Add the \jdbc-odbc\classes; path to your PATH environment variable.
2. Add the \jdbc-odbc\classes; path to your CLASSPATH environment variable.
3. Under Windows 95, put these into your autoexec.bat file and reboot so they take effect for all tasks.
4. Under Windows NT, add these to your Environment tab of your System object in the Control panel, and log off and on again for them to take effect.
Types of JDBC Drivers

There are actually four ways for a Java program to connect to a database:

1. JDBC-ODBC bridge and an ODBC driver -- In this case, it is a local solution, since the ODBC driver and the bridge code must be present on each user's machine. This is fundamentally an interim solution.
2. Native code plus Java driver -- This replaces ODBC and the bridge with another local solution: native code for that platform that can be called by Java.
3. JDBC-Net pure Java driver -- The Java driver-translated JDBC calls into an independent protocol that is passed to a server. The server can then connect to any of a number of databases. This method allows you to call a server from an applet on your client machine and return the results to your applet. In this case, middleware software providers can provide the server.
4. Native protocol Java driver -- The Java driver translates directly to the protocol for that database and makes the calls. This method also can be used over a network and can then display results in a Web browser applet. In this case, each database vendor would supply the driver.

If you want to write code to manipulate a PC client database like dBase, Foxbase, or Access, you will probably use method 1 and have all the code on the user's machine. Larger client-server database products, like IBM's DB2, already provide level 3 drivers for you.

Two-Tier and Three-Tier Models

When the database and the application which interrogates it are on the same machine and there is no server code interposed, we refer to the resulting program as a two-tier model. One tier is the application and the other is the database. This is usually the case in JDBC-ODBC bridge systems.

When there is an application or applet that calls a server that in turn calls the database, we call this a three-tier model. This is usually the case when there is a program called a "server" that you make your calls to.

Writing JDBC Code to Access Databases

Now we are going to start looking into how you write the Java programs themselves to access databases. The database we are going to use is a Microsoft Access database called groceries.mdb. The data in this database consist of the prices for some common commodities at three local grocery stores. The food table looks like this:

<table>
<thead>
<tr>
<th>FoodKey</th>
<th>FoodName</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Apples</td>
</tr>
<tr>
<td>2</td>
<td>Oranges</td>
</tr>
<tr>
<td>3</td>
<td>Hamburger</td>
</tr>
<tr>
<td>4</td>
<td>Butter</td>
</tr>
<tr>
<td>5</td>
<td>Milk</td>
</tr>
<tr>
<td>6</td>
<td>Cola</td>
</tr>
<tr>
<td>7</td>
<td>Green beans</td>
</tr>
</tbody>
</table>

The store table like this:

<table>
<thead>
<tr>
<th>StoreKey</th>
<th>StoreName</th>
</tr>
</thead>
</table>
And the grocery-store pricing table simply consists of keys from these tables followed by prices:

<table>
<thead>
<tr>
<th>FSKey</th>
<th>StoreKey</th>
<th>FoodKey</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>$0.27</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
<td>$0.29</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>1</td>
<td>$0.33</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>2</td>
<td>$0.36</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>2</td>
<td>$0.29</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>2</td>
<td>$0.47</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>3</td>
<td>$1.98</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>3</td>
<td>$2.45</td>
</tr>
<tr>
<td>9</td>
<td>3</td>
<td>3</td>
<td>$2.29</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>4</td>
<td>$2.39</td>
</tr>
<tr>
<td>11</td>
<td>2</td>
<td>4</td>
<td>$2.99</td>
</tr>
<tr>
<td>12</td>
<td>3</td>
<td>4</td>
<td>$3.29</td>
</tr>
<tr>
<td>13</td>
<td>1</td>
<td>5</td>
<td>$1.98</td>
</tr>
<tr>
<td>14</td>
<td>2</td>
<td>5</td>
<td>$1.79</td>
</tr>
<tr>
<td>15</td>
<td>3</td>
<td>5</td>
<td>$1.89</td>
</tr>
<tr>
<td>16</td>
<td>1</td>
<td>6</td>
<td>$2.65</td>
</tr>
<tr>
<td>17</td>
<td>2</td>
<td>6</td>
<td>$3.79</td>
</tr>
<tr>
<td>18</td>
<td>3</td>
<td>6</td>
<td>$2.99</td>
</tr>
<tr>
<td>19</td>
<td>1</td>
<td>7</td>
<td>$2.29</td>
</tr>
<tr>
<td>20</td>
<td>2</td>
<td>7</td>
<td>$2.19</td>
</tr>
<tr>
<td>21</td>
<td>3</td>
<td>7</td>
<td>$1.99</td>
</tr>
</tbody>
</table>

Registering Your Database with ODBC

Before you can access an ODBC database under Windows 95 or NT, you must register it with the ODBC driver control panel. Under Windows 95, this is the ODBC icon in the Control Panel program. Under Windows NT, you will find the program on the Start menu. (If you don't, you will need to install the ODBC driver WX1350.exe we described above.)

Double-click on the ODBC icon and then click on Add as shown in Figure 1. Then select a database driver (here use Microsoft Access) and click on OK. Type in a Data Source name (Grocery prices) and a Description (in local stores) for the database (neither of these need be related to the filename), and then click on Select to locate the database and select it. After locating the database, your screen will appear as shown in Figure 2. Click on OK and then on Close to close the panels.

**Figure 1: The ODBC control panel setup screen**

Stop and Shop
Village Market
Waldbaum's
Connecting to a Database

All of the database objects and methods are in the java.sql package, and you must import java.sql.* into any programs that use JDBC. To connect to an ODBC database, you must first load the JDBC-ODBC bridge driver

```java
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

String url = "jdbc:odbc:Grocery prices";
Connection con = DriverManager.getConnection(url);
```
Accessing the Database

Once you have connected to the database, you can request information on the names of the tables and on the names and contents of their columns, and you can run SQL statements that either query the database or add to or modify its contents. The objects that you can use to obtain information from the database are:

**DatabaseMetaData**
Information about the database as a whole: table names, table indexes, database product name and version, actions supported by database.

**ResultSet**
Information about a table or result of a query. You access the data row by row, but you can access the columns in any order.

**ResultSetMetaData**
Information about the column names and types in a ResultSet.

While each of these objects has a large number of methods that allow you to get very detailed information about the elements of the database, there are a few major methods in each object that give you the most significant information about your data. However, if you are looking for more information than we show here, we encourage you to study the documentation for descriptions of the remaining methods.

The ResultSet

The ResultSet object is the most important single object in JDBC. It is essentially an abstraction of a table of general width and unknown length. Nearly all methods and queries return data as a ResultSet. A ResultSet contains any number of named columns that you can ask for by name. It also consists of one or many rows, which you can move through sequentially from top to bottom one at a time. Before you can use a ResultSet, you need to ask how many columns it contains. This information is stored in the ResultSetMetaData object.

```java
//get the number of columns from the metadata
ResultSetMetaData rsmd;
rsmd = results.getMetaData();
numCols = rsmd.getColumnCount();
```

When you obtain a ResultSet, it points just before the first row. You use the next() method to obtain each additional row, and the method returns false when no more rows remain. Since fetching data from a database may result in errors, you must always enclose your result set manipulations in a try block.

```java
try
```
You can fetch data in a ResultSet in many forms, depending on the data type stored in each column. Further, you can obtain the contents of a column either by column number or by column name. Note that column numbers start at 1, not at 0. Some of the more common methods for the ResultSet object are shown below.

```java
rsmd = results.getMetaData();
numCols = rsmd.getColumnCount();
boolean more = results.next();
{
    for (i = 1; i <= numCols; i++)
        System.out.print(results.getString(i) + " ");
    System.out.println();
    more = results.next();
} results.close();
} catch(Exception e)
{ System.out.println(e.getMessage()); }
```

### getInt(int);  
Returns contents of the column numbered int as an integer.

### getInt(String);  
Returns contents of the column named String as an integer.

### getFloat(int);  
Return contents of the column numbered int as a float.

### getFloat(String);  
Returns contents of the column named String as a float.

### getDate(int);  
Return contents of the column numbered int as a date.

### getDate(String);  
Returns contents of the column named String as a date.

### next();  
Moves the row pointer to the next row. Returns false if no rows remain.

### close();  
Closes the result set.

### getMetaData();  
Returns the ResultSetMetaData object.

---

### ResultSetMetaData

You obtain the ResultSetMetaData object from the ResultSet using the getMetaData() method. You can use this object to discover the number and type of columns and the names of each column.

```java
getColumnCount(); Returns the number of columns in the ResultSet.
getColumnName(int); Returns the name of column number int.
getColumnLabel(int); Returns the suggested label for that column.
isCurrency(int); Returns true if this column contains a number in currency units.
isReadOnly(int); Returns true if the column is read only.
isAutoIncrement(int); Returns true if this column is autoincrement.
```
The `DatabaseMetaData` object gives you information about the entire database. You use it primarily to get the names of the tables in a database and the names of the columns in a table. Since various databases also support different variants of SQL, there are also a large number of methods querying the database about what SQL methods it supports.

### DatabaseMetaData

The `DatabaseMetaData` object gives you information about the entire database. You use it primarily to get the names of the tables in a database and the names of the columns in a table. Since various databases also support different variants of SQL, there are also a large number of methods querying the database about what SQL methods it supports.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>getCatalogs()</code></td>
<td>Return a list of catalogs of information in that database. With the JDBC-ODBC Bridge driver, you get a list of databases registered with ODBC. This is seldom used in JDBC-ODBC databases.</td>
</tr>
<tr>
<td><code>getTables(catalog, schema, tableNames, columnNames)</code></td>
<td>Returns a description of the table names for all tables matching tableNames and all columns matching columnNames.</td>
</tr>
<tr>
<td><code>getColumns(catalog, schema, tableNames, columnNames)</code></td>
<td>Returns a description of the table column names for all tables matching tableNames and all columns matching columnNames.</td>
</tr>
<tr>
<td><code>getURL()</code></td>
<td>Get the name of the URL you are connected to.</td>
</tr>
<tr>
<td><code>getDriverName()</code></td>
<td>Get the name of the database driver you are connected to.</td>
</tr>
</tbody>
</table>

### Getting Information on Tables

- **catalog**: The name of the catalog to look in for table names. For JDBC-ODBC databases, and many others, this can be set to null. The catalog entry for these databases is actually their...
absolute pathname in the file system.

schema The database "schema" to include. Many databases do not support schema, and for others it is the user name of the owner of the database. It is usually set to null.

tablemask A mask describing the names of the tables you want to retrieve. If you want to retrieve all table names, set it to the wildcard character %. Note that the wildcard character in SQL is the %-sign and not the usual PC user's *-sign.

types[] This is an array of Strings describing the kinds of tables you want to retrieve. Databases frequently contain a number of tables for internal housekeeping that are of little value to you as a user. If this is null, you will get all these tables. If you make this a one-element array containing the string "TABLES", you will get only the tables of interest to users.

Simple code for getting the table names in a database amounts to getting the DatabaseMetaData object and retrieving the table names from it:

```java
con = DriverManager.getConnection(url);
//get the database metadata
dma = con.getMetaData();
//now dump out the names of the tables in the database
String[] types = new String[1];
types[0] = "TABLES"; //set table type mask
//note the %-sign is a wild card (not '*')
results = dma.getTables(null, null, "%", types);

boolean more = results.next();
while (more)
{ 
  for (i = 1; i <= numCols; i++)
    System.out.print(results.getString(i)+"   ");
  System.out.println();
  more = results.next();
}
```

Then we can print out the table names, just as we did above:

```java
boolean more = results.next();
while (more)
{ 
  for (i = 1; i <= numCols; i++)
    System.out.print(results.getString(i)+"   ");
  System.out.println();
  more = results.next();
}
```

Enclose all the code in a try block as before.

---

**Executing SQL Queries**

---

**Printing out ResultSets**

---

**A Simple JDBC Program**

```
C:\Projects\objectJava\chapter19> java JdbcOdbc_test
```
Building Higher Level JDBC Objects

It is plain from the foregoing example that it would be very helpful if we could encapsulate some of the behavior we have been using in a few higher-level objects. Not only could we enclose the try blocks, but we could make access to the ResultSet methods a little simpler.

In this section, we'll build a new ResultSet object which encapsulates the JDBC ResultSet object and returns the data in a row as an array of Strings. We discovered that you always needed to obtain the number of columns and the names of the columns from the ResultSetMetaData object, and thus it seems quite reasonable to create a new object that also encapsulates the metadata.

class resultSet
{
    // this class is a higher level abstraction
    // of the JDBC ResultSet object
    ResultSet rs;
    ResultSetMetaData rsmd;
    int numCols;
    public resultSet(ResultSet rset)
    {
        rs = rset;
        try
        {
            // get the meta data and column count at once
            rsmd = rs.getMetaData();
            numCols = rsmd.getColumnCount();
        }
        catch (Exception e)
        {
            System.out.println("resultset error"
                              + e.getMessage());
        }
    }
    //--
    public String[] getMetaData()
    {
        // returns an array of all the column names
        // or other meta data
        String md[] = new String[numCols];
        try
        {
            for (int i=1; i<= numCols; i++)
            {
                md[i-1] = rsmd.getColumnName(i);
            }
        }
        catch (Exception e)
        {
            System.out.println("meta data error" +
                                e.getMessage());
        }
    }
}
Building a Database Object

ResultSet results = .. //get a ResultSet as usual
//create this more useful object from it
resultSet rs = new resultSet(results);

Using JDBC to create database objects

A Visual Database Program

To wrap up the material we've covered in this chapter, let's write a simple GUI program which allows us to display the table names, column names, and column contents of a database. We'll also include a text area where you can type in an SQL query to execute on that Database. You will find the ResultSet and Database classes used in this program, called dbFrame.java, in the \chapter20 subdirectory on the Companion CD-ROM. The display of the program is shown in Figure 3.

Figure 3: The dbFrame.java program for displaying data in a JDBC-connected database

In this program, the table names of our default database, groceries.mdb, are shown in the left column. When you click on one of the table names, the column names are shown in the middle column. Finally, when you click on a row in the middle column, the contents of that column are shown in the right-hand column.

```java
public void itemStateChanged(ItemEvent e) {
    Object obj = e.getSource();
    if (obj == Tables)     //put in column names
        showColumns();
    if (obj == Columns)    //put in column contents
        showData();
}

//--
private void loadList(List list, String[] s) {
    //clear and fill specified list box
    list.removeAll();
    for (int i=0; i< s.length; i++)
        list.add(s[i]);
}

//--
private void showColumns() {
    //display the column names
    String cnames[] =
        db.getColumnNames(Tables.getSelectedItem());
    loadList(Columns, cnames);
}

//--
private void showData() {
    String colname = Columns.getSelectedItem();
    String colval =
        db.getColumnValue(Tables.getSelectedItem(),
                         colname);
```
Executing a Query

public void actionPerformed(ActionEvent e) {
    Object obj = e.getSource();
    if (obj == Quit)
        System.exit(0);
    if (obj == Search)
        clickedSearch();
}

private void clickedSearch() {
    ResultSet rs = db.Execute(query.getText());
    String cnames[] = rs.getMetaData();
    queryDialog q = new queryDialog(this, rs);
    q.show();
}

The Query Result Dialog

public void paint(Graphics g) {
    String s[];
    int x=0;
    //compute the font height
    int y = g.getFontMetrics().getHeight();
    //compute an estimated column width
    int deltaX = (int)1.5f *
        (g.getFontMetrics().stringWidth("wwwwwwwwwwwwwwwwww"));
    //move through the table vector
    for (int i=0; i<tables.size(); i++)
    {
        s  = (String[])tables.elementAt(i);
        //and draw each row from the string array
        for (int j =0; j< s.length; j++)
        {
            String st= s[j];
            g.drawString(st, x, y);
            x += deltaX;           //move over to next column
        }
        x = 0;                 //start a new row
        y += g.getFontMetrics().getHeight();
          ///extra space between column labels and their data
        if (i == 0) y += g.getFontMetrics().getHeight();
    }
}
Figure 4: The queryDialog display in the dbFrame program, showing the results of the default query

Example Files

groceries.zip
dbFrame.zip
jdbc-odbc Bridge

Summary

In this article, we've discussed databases and ways of examining them and executing queries on them. We've seen that the JDBC provides a platform and a database-independent, object-oriented way to access these data, and we've looked at the major objects of JDBC: the ResultSet, ResultSetMetaData, and DatabaseMetaData. After writing a simple program using these objects, we designed higher-level resultSet and Database objects which we used to build a simple visual interface to display database information.

If you are familiar with the power of databases, you will realize that the SQL language allows you to carry out many more powerful operations than we've summarized here. For example, you could create new tables, add columns, and add, change, or delete rows, columns, or individual cells of a table. All of this becomes quite general and easy to manipulate using JDBC.

If you are using a platform-specific database driver like the JDBC-ODBC Bridge, you are restricted to writing applications, since applets cannot connect to this bridge running on another computer. Other client-server databases, like IBM's DB2, do allow you to connect to them using an applet JDBC.

About the author

James W. Cooper works at the IBM T.J. Watson Research Center