**8. Java Database Connectivity**

**Outline**

* JDBC Overview
* Introduction to SQL
* Database and Driver Types
* JDBC Supporting Classes
* Procedures for using JDBC to Access Database
* Data Source Administrator
* Java Classes used in the Examples
* Connect to Database and Make a Simple Query: Example Fig. 18.24: TableDisplay.java
* Display Query Results Data: Example Fig. 18.29: Querying the Books.mdb Database
* Example Fig. 18.30: Reading, Inserting, and Updating a Microsoft Access Database

**What is JDBC?**

* A set of SQL-Level APIs to enable Java applications to access relational database
* Java program will be able to use SQL(Structure Query Language) query statements to access information in relational database systems

**An Introduction to SQL**

SQL Commands

* SELECT
* INSERT
* UPDATE
* DELETE

**CREATE** TABLE tableName(

columnName columnType columnModifier,

columnName columnType columnModifier,

…

columnName columnType columnModifier,

)

**INSERT** INTO tableName(cloumnName, …, columnName)

VALUES(value, value, .., value)

**UPDATE** tableName

SET columnName = value,

columnName = value,

….

columnName = value,

WHERE columnName = value,

**DELETE** FROM tableName WHERE columnName = value

**SELECT** columnName, .. , columnName

FROM tableName

WHERE columnName = value

**Some SQL Query Examples**

SELEC \* FROM Atthors WHERE YearBorn > 1960

SELECT AuthorID, LastName FROM Authors

SELECT \* FROM Authors WHERE LastName LIKE 'd\*'

SELECT \* FROM Authors WHERE LastName LIKE '?\*'

SELECT \* FROM Authors WHERE LastName LIKE '?[a-I]\*'

**ORDER BY CLAUSE**

SELECT \* FROM Authors ORDER BY LastName ASC

SELECT \* FROM Authors ORDER BY LastName DESC

SELECT \* FROM Authors ORDER BY LastName, FirstName

SELECT \* FROM Titles

WHERE Title LIKE '\*How to Program'

ORDER BY Title ASC

INNER JOIN

SELECT \* FROM Tables WHERE JOIN Table2 ON Table1.field = Table2.field

SELECT FirstName, LastName, ISBN

FROM Authors INNER JOIN AuthorISBN

ON Authors.AuthordID = AuthorISBN.AuthorID

ORDER BY LastName, FirstName

**SQL Examples:**

TitleAuthor Query from Books.mbd

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

**Database and Driver Types**

Type 1 Driver (**JDBC-ODBC bridge**)

* A bridge technology that provides a gateway to the ODBC API (JDBC-ODBC)
* Comes with JDK for accessing a database
* Requires software installation on client side

Type 2 Driver (**Native API connection driver**)

* Native API connection driver contains Java code that calls native C or C++ methods
* Provided by the individual database vendors
* Require software installation on client side

Type 3 Driver (**Network Connection Driver**)

* JDBC driver on client side
* Uses sockets to call a middleware application on the server
* The middleware translates the client request
* No software installation is required

Type 4 Driver (**Database Protocol Driver**)

* Pure Java solution using network protocols built into the database engine
* Using Java socket to talk directly to the database

JDBC Access Driver API Web site:

http://industry.java.sun.com/products/jdbc/drivers

**JDBC Supporting Classes**

* The JDBC Classes for Creating a Connection

java.sql.Driver

java.sql.DriveManager

java.sql.Connection

* Database Access

java.sql.Statement

// executeQuery() to execute SQL query

// executeUpdate()

java.sql.ResultSet

// one or more rows of data returned from a database query

* Other classes
  + java.sql.SQLException
  + java.sql.SQLWarning
  + java.sql.DataTruncation
  + java.sql.Date
  + java.sql.Time
  + java.sql.Timestamp
  + SQL Datatypes and Java Datatypes Mappings

**java.sql.Types Java Type**

BIT boolean

TINYINT byte

SMALLINT short

INTEGER int

BIGINT long

REAL float

FLOAT double

DOUBLE double

**java.sql.Types Java Type**

DECIMAL java.math.BigDecimal

NUMERIC java.math.BigDecimal

CHAR java.lang.String

VARCHAR java.lang.String

LONGCARCHAR java.lang.String

DATE java.sql.Date

TIME java.sql.Time

TIMESTAMP java.sql.Timestamp

BINARY byte[ ]

VARBINARY byte[ ]

LONGVARBINARY byte[ ]

BLOB java.sql.Blob

CLOB java.sql.Clob

ARRAY java.sql.Array

REF java.sql.Ref

STRUCT java.sql.Struct

**Procedures for using JDBC to Access Database**

The following steps are necessary to access a database using JDBC:

* Preparation
  + Install Java and JDBC API
  + Install a driver
* Setting up a database
* Establishing a connection with the DBMS
  + Loading a driver and register it with java.sql.DriverManager
    - If using the JDBC-ODBC bridge driver:

**Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");**

* + - If using JDBC driver:

**Class.forName("jdbc.DriverXYZ");**

* + Open a connection

// ODBC Data source called "ThisDb"

String url = "jdbc:odbc:ThisBookTable";

String dbUserName = "anonymous";

String dbPassword = "guest";

connection conDB = DriverManager.getConnection(url, dbUserName, dbPassword);

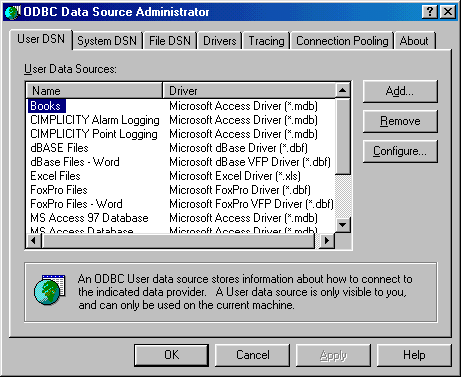
* Using JDBC Statements
  + Using a **Statement** object to send SQL statements to the DBMS

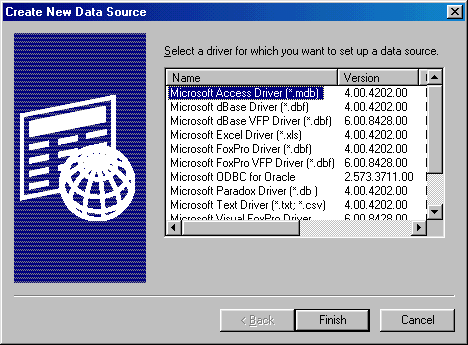
**Statement dbStmExec = conDB.createsStatement();**

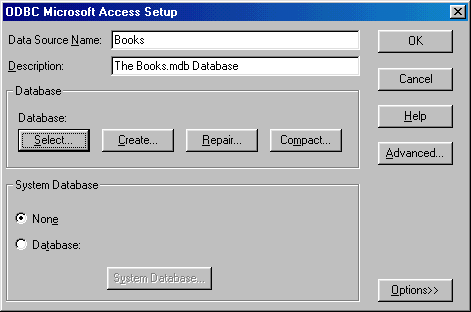
**dbStmExec.executeUpdate(sqlQueryString);**

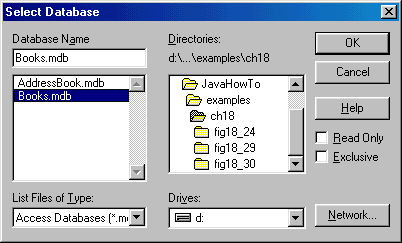
**Data Source Administrator**

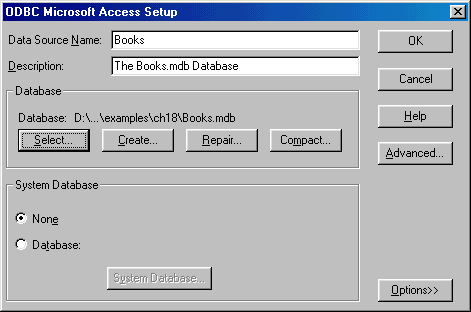
**Configuring ODBC Data Source (Window's control panel)**

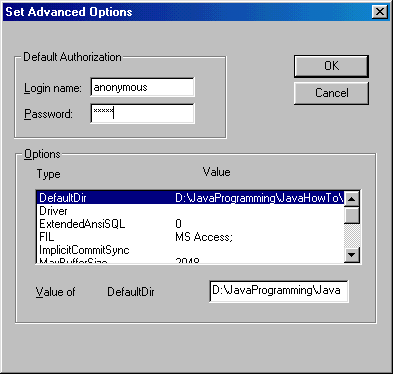










****

**Java Classes Used in the JDBC Examples**

**java.lang.Class**

**public final class Class extends Object implements Serializable**

* Instances of the class Class represent classes and interfaces in a running Java application.
* Every array also belongs to a class that is reflected as a Class object that is shared by all arrays with the same element type and number of dimensions.
* The primitive Java types (boolean, byte, char, short, int, long, float, and double), and the keyword void are also represented as Class objects.

Static Class forName(String className)

* Create a new Class object associated with the class or interface with the given string name.

**java.sql**

**public interface connection**

* An interface for creating a connection (session) with a specific database
* Within the context of a Connection, SQL statements are executed and results are returned.

A Connection's database is able to provide information describing its tables, its supported SQL grammar, its stored procedures, the capabilities of this connection, and so on. This information is obtained with the getMetaData method.

**java.sql**

**public interface Statement**

* The object used for executing a static SQL statement and obtaining the results produced by it.
* Only one ResultSet object per Statement object can be open at any point in time.
* If the reading of one ResultSet object is interleaved with the reading of another, each must have been generated by different Statement objects. All statement execute methods implicitly close a statment's current ResultSet object if an open one exists.

**java.sql**

**public interface ResultSet**

The ResultSet object references to a table of data representing a database result set. The ResultSet object is usually generated by executing a statement that queries the database. A ResultSet object maintains a cursor pointing to its current row of data. Initially the cursor is positioned before the first row. The next method moves the cursor to the next row, and because it returns false when there are no more rows in the ResultSet object, it can be used in a while loop to iterate through the result set.

**Fields and Methods of ResultSet**

// Fields

static int CONCUR\_READ\_ONLY

static int CONCUR\_UPDATABLE

static int FETCH\_FORWARD

static int FETCH\_REVERSE

static int FETCH\_UNKNOWN

static int TYPE\_FORWARD\_ONLY

static int TYPE\_SCROLL\_INSENTIVE

static int TYPE\_SCROLL\_SENSITIVE

//Methods

boolean absolute(int row)

void afterLast()

void beforeFirst()

void cancelRowUpdates()

void clearWarnings()

void close()

void deleteRow()

int findColumn(String columName)

boolean first()

Array getArray(int I)

Array getArray(String colName)

InputStream getAsciiStream(int columnIndex)

InputStream getAsciiStream(String columnName)

BigDecimal getBigDecimal(int columnIndex)

BigDecimal getBigDecimal(int columnIndex, int scale)

BigDecimal getBigDecimal(String columnName)

BigDecimal getBigDecimal(String columnName, int scale)

InputStream getBinaryStream(int columnIndex)

InputStream getBinatyStream(String columnName)

Blob getBlob(int I)

Blob getBlob(String colName)

boolean getBoolean(int columnIndex)

boolean getBoolean(String columnName)

byte getByte(int columnIndex)

byte[ ] getByte(String columnName)

byte[ ] getBytes(int columnIndex)

byte[ ] getByte(String columnName)

Reader getCharacterStream(int columnIndex)

Reader getCharacterStream(String columnName)

Clob getClob(int i)

Clob getClob(String colName)

int getConcurrency()

String getCursorName()

Date getDate(int columnIndex)

Date getDate(int columnIndex, Calendar cal)

Date getDate(String columnName)

Date getDate(String columnName, Calendar cal)

double getDouble(int columnIndex)

double getDouble(String columnName)

int getFethcDirection()

int getFetchSize()

float getFloat(int columnIndex)

float getFloat(Sting columnName)

int getInt(int columnIndex)

int getInt(String columnName)

long getLong(int columnIndex)

long getLong(String columnName)

ResultSetMetaData getMetaData()

Object getObject()

Object getObject(int i, Map map)

Object getObject(String columnName)

Object getObject(String colName, Map map)

Ref getRef(int i)

Ref getRef(String colName)

int getRow()

short getShort(int columnIndex)

short getShort(String columnName)

Statement getStatement()

String getString(int columnIndex)

String getString(String columnName)

Time getTIme(int columnIndex)

Time getTime(int columnIndex, Calendar cal)

Time getTime(String columnName)

Time getTime(String columnName, Calendar cal)

Timestamp getTimestamp(int columnIndex)

Timestamp getTimestamp(int columnIndex, Calendar cal)

Timestamp getTimestamp(String columnName)

Timestamp getTimestamp(String columnName, Calendar cal)

int getType()

InputStream getUnicodeStream(int columnIndex)

InputStream getUnicodeStream(String columnName)

SQLQarning getWarnings()

void insertRow()

boolean isAfterLast()

boolean isBeforeFirst()

boolean isFirst()

boolean isLast()

boolean last()

void moveToCurrentRow()

void moveToInsertRow()

boolean next()

boolean previous()

void refreshRow()

boolean relative(int rows)

boolean rowDeleted()

boolean rowInserted()

boolean rowUpdate()

void setFetchDirection(int direction)

void setFetchSize(int rows)

void updateAsciiStream(int columnIndex, InputStream x, int length)

void updateAsciiStream(String columnName, InputStream x, int length)

void updateBigDecimal(int columnIndex, BigDecimal x)

void updateBigDecimal(String columnName, BigDecimal x)

void updateBinaryStream(int columnIndex, InputStream x, int length)

void updateBinaryStream(String columnName, InputStream x, int length)

void updateBoolean(int columnIndex, boolean x)

void updateBinaryStream(String columnName, boolean x)

void updateByte(int columnIndex, byte x)

void updateByte(String columnName, byte x)

void updateByte(String columnName, byte [ ] x)

void updateBytes(String columnName, byte [ ] x)

void updateCharacterStream(int columnIndex, Reader x, int length)

void updateCharacterStream(String columnName, Reader x, int length)

void updateDate(int columnIndex, Date x)

void updateDate(String columnName, Date x)

void updateDouble(int columnIndex, double x)

void updateDouble(Sting columnName, double x)

void updateFloat(int columnIndex, Float x)

void updateFloat(Sting columnName, Float x)

void updateInt(int columnIndex, int x)

void updateInt(Sting columnName, int x)

void updateLong(int columnIndex, long x)

void updateLong(Sting columnName, long x)

void updateNull(int columnIndex)

void updateNull(Sting columnName)

void updateObject(int columnIndex, Objec x)

void updateObject(int columnIndex, Object x, int scale)

void updateObject(Sting columnName, Object x)

void updateObject(Sting columnName, Object x, int scale)

void updateRow()

void updateShort(int columnIndex, Short x)

void updateShort(Sting columnName, Short x)

void updateString(int columnIndex, String x)

void updateString(Sting columnName, String x)

void updateTime(int columnName, Time x)

void updateTime(String columnName, Time x)

void updateTimestamp(int columnName, Timestamp x)

void updateTimestamp(String columnName, Timestamp x)

boolean wasNull()

**Other Classes referenced by ResultSet**

public **class Date** extends Object implements Serializable, Cloneable, Comparable

// Give the information on Year, Month, Day, Hour, Minute, and Second

// value

public **class Time** extends Date

// A thins wrapper class extends java.util.Date

// Allows the JDBC API to identify this as an SQL TIME values

public **class Timestamp** extends Date

// A thins wrapper class extends java.util.Date

// Allows the JDBC API to identify this as an SQL TIMESAMP value

public abstract **class Calendar** extends Object implements Serializable, Clonable

// A base class for converting between a Date object and a set of

// integer fields: YEAR, MONTH, DAY, HOUR, etc

public **interface Ref**

// The Ref is a reference to a SQL structured type value in the database

public **interface Map**

// Support mapping mechanism from key to value

public **interface Clob**

// A SQL Clob type mapping support

// A SQL CLOB is a built-in type that stores a Character Large Object as

// column value in a row of a database table

public **interface Blob**

// A SQL Blob type mapping support

// A SQL BLOB is a built-in type that stores a Binary Large Object as

// column value in a row of a database table

public **class BigDecimal** extends Number implements Comparable

// Arbitrary-precision signed decimal numbers

public **abstract class Reader** extends Object

// An abstract class for reading character streams

public **abstract class Writer** extends Object

// An abstract class for writing character streams

**java.sql**

**public interface ResultSetMetaData**

This interface is designed with fields and methods for retrieving and determining information about columns of database data being placed in ResltSet object through an execution of execQuery() method.

* This interface enables an object to get information about the types and properties of the columns in a ResultSet object.
* An example:
  + **ResultSet object rs** is created
  + Execute query method **ResultSet rs** to receive the database information
  + The **ResultSetMetaData rsmd** object can be created
  + Find out how many columns rs has

String query = "SELECT \* FROM Authors";

ResultSet rs = **stmt.executeQuery(query")**;

ResultSetMetaData rsmd = **rs.getMetaData()**;

int numberOfColumns = **rsmd.getColumnCount()**;

* + Determine if the 1st column can be used in a WHERE clause

String query = "SELECT \* FROM Authors";

ResultSet rs = stmt.executeQuery(query");

ResultSetMetaData rsmd = rs.getMetaData();

int numberOfColumns = rsmd.getColumnCount();

boolean b = **rsmd.isSearchable(1)**;

**Fields and Methods of ResultSetMetaData**

//Fields

static int columnNoNulls // A column does not allow NULL values

static int columnNullable // A column doe allow NULL values

static int columnNullableUnknown // Column nullability is unknown

//Methods

String getCatalogName(int column) throws SQLException

// Gets the designated column's table's catalog name

String getColumnClassname(int column) throws SQLException

// Returns the fully qualified name of the Java class whose instances are int getColumnCount() throws SQLException

// Returns the numbers of column in ResultSet Object

// throws SQLException If a database access error occur

int getColumnDisplaySize(int column) throws SQLException

// Determines the max width in characters for the selected

// column

String getColumnLabel(int column) throws SQLException

// Retrieve TITLE from selected column for display or pritining

String getColumnName(int column) throws SQLException

// Retrieve the selected column name

int getColumnType(int column) throws SQLException

// Retrieve the designated column's SQL type

int getColumnTypeName(int column) throws SQLException

// Retrieve the selected column type name

int getPrecision(int column) throws SQLException

// Get information about the number of decimal digits for the

// selected column

int getScale(int column) throws SQLException

// Get the designated column's number of digits to right of decimal points

String getSchemaName(int column) throws SQLException

// Get the designated column table schema

String getTableName(int column) throws SQLException

// Get the designated column's table name

boolean isAutoIncreament(int column) throws SQLException

// Indicated if the designated column is automatically numbered

// First column = 1, second column = 2, etc

boolean isCaseSensitive(int column) throws SQLException

boolean isCurrency(int column) throws SQLException

// Determine if the column is currency value

boolean isDefinitelyWritable(int column) throws SQLException

// Determine if the column is definitely writable

int isNullable(int column) throws SQLException

boolean isReadOnly(int column) throws SQLException

// Determine if the column is read only

boolean isSearchable(int column) throws SQLException

// Indicates whether the designated columns can be used in

// a SQL WHERE cluase

boolean isSigned(int column) throws SQLException

// Determine if the values in the designated columns are signed numbers

boolean isWriteable(int column) throws SQLException

// Determine if the column is writable

**java.sql**

**public interface SQLData**

* The interface used for the custom mapping of SQL user-defined types
* This interface must be implemented by any Java class that is registered in a type mapping.
* It is expected that this interface will normally be implemented by a tool.
* The methods in this interface are called by the driver and are never called by a programmer directly.

**java.sql**

**public interface SQLInput**

* An input stream that contains a stream of values representing an instance of an SQL structured or distinct type.
* This interface, used only for custom mapping, is used by the driver behind the scenes, and a programmer never directly invokes SQLInput methods. The readXXX methods provide a way to read the values in an SQLInput object. The method wasNull is used to determine whether the the last value read was SQL NULL.
* When the method getObject is called with an object of a class implementing the interface SQLData, the JDBC driver calls the method SQLData.getSQLType to determine the SQL type of the user-defined type (UDT) being custom mapped.
* The driver creates an instance of SQLInput, populating it with the attributes of the UDT. The driver then passes the input stream to the method SQLData.readSQL, which in turn calls the SQLInput.readXXX methods in its implementation for reading the attributes from the input stream.

**public interface SQLOutput**

* The output stream for writing the attributes of a user-defined type back to the database. This interface, used only for custom mapping, is used by the driver, and its methods are never directly invoked by a programmer.
* When an object of a class implementing the interface SQLData is passed as an argument to an SQL statement, the JDBC driver calls the method SQLData.getSQLType to determine the kind of SQL datum being passed to the database. The driver then creates an instance of SQLOutput and passes it to the method SQLData.writeSQL. The method writeSQL in turn calls the appropriate SQLOutput.writeXXX methods to write data from the SQLData object to the SQLOutput output stream as the representation of an SQL user-defined type.

**public class SQLException**

**extends** [**Exception**](C:jdk1_3_0_02jdk1.3docsapijavalangException.html)

An exception that provides information on a database access error or other errors.

Each SQLException provides several kinds of information:

* a string describing the error. This is used as the Java Exception message, available via the method getMesage.
* a "SQLstate" string, which follows the XOPEN SQLstate conventions. The values of the SQLState string are described in the XOPEN SQL spec.
* an integer error code that is specific to each vendor. Normally this will be the actual error code returned by the underlying database.
* a chain to a next Exception. This can be used to provide additional error information

**java.sql**

**public class DriveManager extends Object**

* This class provides basic service for managing a set of JDBC drivers
* During the initialization through the method getConnection, the DriverManager class attempts to load the driver classes referenced in the "jdbc.drivers" system property
* A program can also explicitly load JDBC drivers at any time. For example, the my.sql.Driver is loaded with the following statement

**NOTE:** The DataSource interface, new in the JDBC 2.0 API, provides another way to connect to a data source. The use of a DataSource object is the preferred means of connecting to a data source.

**public class JTable**

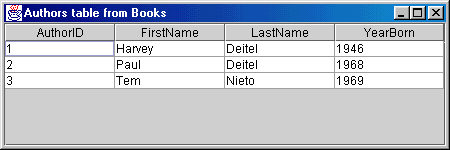
extends [JComponent](C:jdk1_3_0_02jdk1.3docsapijavaxswingJComponent.html)

implements [TableModelListener](C:jdk1_3_0_02jdk1.3docsapijavaxswingeventTableModelListener.html), [Scrollable](C:jdk1_3_0_02jdk1.3docsapijavaxswingScrollable.html), [TableColumnModelListener](C:jdk1_3_0_02jdk1.3docsapijavaxswingeventTableColumnModelListener.html), [ListSelectionListener](C:jdk1_3_0_02jdk1.3docsapijavaxswingeventListSelectionListener.html), [CellEditorListener](C:jdk1_3_0_02jdk1.3docsapijavaxswingeventCellEditorListener.html), [Accessible](C:jdk1_3_0_02jdk1.3docsapijavaxaccessibilityAccessible.html)

2D table user-interface component

**Example 1: Connect to database and make a simple query.**

**Fig.18.24: TableDisplay.java**



// Fig. 18.24: TableDisplay.java

// This program displays the contents of the Authors table

// in the Books database.

import java.sql.\*;

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.\*;

import java.util.\*;

public class TableDisplay extends JFrame {

**private Connection connection;**

**private JTable table;**

public TableDisplay()

{

// The URL specifying the Books database to which

// this program connects using JDBC to connect to a

// Microsoft ODBC database.

String url = "jdbc:odbc:Books";

String username = "anonymous";

String password = "guest";

// Load the driver to allow connection to the database

try {

**Class.forName( "sun.jdbc.odbc.JdbcOdbcDriver" );**

**connection = DriverManager.getConnection(**

**url, username, password );**

}

catch ( **ClassNotFoundException cnfex** ) {

System.err.println(

"Failed to load JDBC/ODBC driver." );

**cnfex.printStackTrace();**

System.exit( 1 ); // terminate program

}

catch ( SQLException sqlex ) {

System.err.println( "Unable to connect" );

sqlex.printStackTrace();

}

getTable();

setSize( 450, 150 );

show();

}

private void getTable()

{

**Statement statement; // Sql Interafce**

**ResultSet resultSet; // Sql Interface**

try {

String query = "SELECT \* FROM Authors";

**statement = connection.createStatement();**

**resultSet = statement.executeQuery( query );**

displayResultSet( resultSet );

**statement.close();**

}

catch ( SQLException sqlex ) {

sqlex.printStackTrace();

}

}

**private void displayResultSet( ResultSet rs )**

throws SQLException

{

// position to first record

**boolean moreRecords = rs.next()**;

// If there are no records, display a message

if ( ! moreRecords ) {

JOptionPane.showMessageDialog( this,

"ResultSet contained no records" );

setTitle( "No records to display" );

return;

}

setTitle( "Authors table from Books" );

**Vector columnHeads = new Vector();**

**Vector rows = new Vector();**

try {

// get column heads

**ResultSetMetaData rsmd = rs.getMetaData();**

for ( int i = 1; i <= **rsmd.getColumnCount();** ++i )

**columnHeads.addElement( rsmd.getColumnName( i ) )**;

// get row data

do {

**rows.addElement( getNextRow( rs, rsmd ) );**

} while ( **rs.next()** );

// display table with ResultSet contents

table = new JTable( rows, columnHeads );

JScrollPane scroller = new JScrollPane( table );

getContentPane().add(

scroller, BorderLayout.CENTER );

validate();

}

catch ( SQLException sqlex ) {

sqlex.printStackTrace();

}

}

private Vector getNextRow( ResultSet rs,

ResultSetMetaData rsmd )

throws SQLException

{

Vector currentRow = new Vector();

for ( int i = 1; i <= **rsmd.getColumnCount()**; ++i )

switch( **rsmd.getColumnType( i )** ) {

case Types.VARCHAR:

**currentRow.addElement( rs.getString( i ) )**;

break;

case Types.INTEGER:

**currentRow.addElement**(

new Long( rs.getLong( i ) ) );

break;

default:

System.out.println( "Type was: " +

**rsmd.getColumnTypeName**( i ) );

}

return currentRow;

}

public void shutDown()

{

try {

connection.close();

}

catch ( SQLException sqlex ) {

System.err.println( "Unable to disconnect" );

sqlex.printStackTrace();

}

}

public static void main( String args[] )

{

final TableDisplay app = new TableDisplay();

app.addWindowListener(

new WindowAdapter() {

public void windowClosing( WindowEvent e )

{

app.shutDown();

System.exit( 0 );

}

}

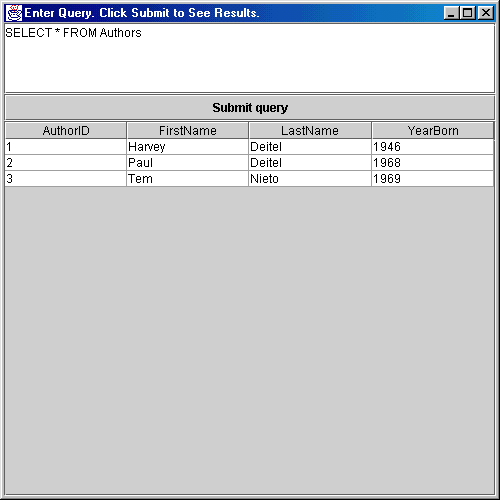
);

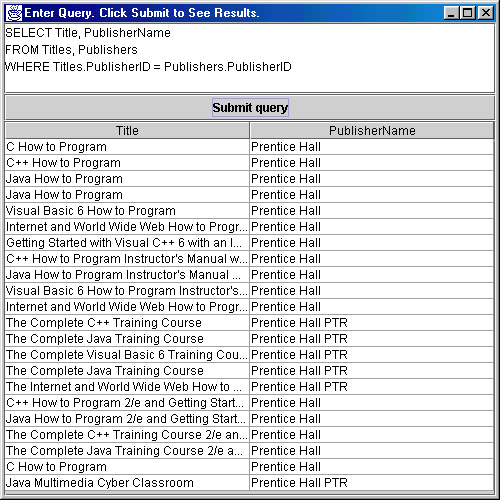
}

}

**Example 2 Querying Books.mdb Database**

**Fig. 18.29: Querying the Books.mdb Database**





**Example 3: Database Reading, Inserting and Updating**

**Fig. 18.30: Reading, Inserting, and Updating an One Table Microsoft Access Database**

