database to java

Week 07

Instructor : Dr. Saif Ur Rehman
Lab Engineer: Muhammad Hassan Mujtaba

Today Contents

- Database Management System (DBMS)
  - Mechanisms for storing and organizing data
  - Access, store, modify data without concern for internal representation (information hiding)
- Structured Query Language (SQL)
  - Standard language used with relational databases to perform queries and manipulate data
- Java Database Connectivity (JDBC)
  - Java programs communicate with databases using JDBC
  - JDBC driver implements interface to database

Introduction

- **Database Management System (DBMS)**
  - Mechanisms for storing and organizing data
  - Access, store, modify data without concern for internal representation (information hiding)
- **Structured Query Language (SQL)**
  - Standard language used with relational databases to perform queries and manipulate data
- **Java Database Connectivity (JDBC)**
  - Java programs communicate with databases using JDBC
  - JDBC driver implements interface to database

Relational-Database Model

- **Relational database**
  - Logical representation of data, not necessarily the way the data is stored
  - Table
    - Rows (entities), columns (attributes)
  - Primary key (column or group of columns)
    - Unique value for each row
    - Not every table has a primary key
- **SQL statement**
  - Query (which data to select from table or tables)

Relational-Database Model (Cont.)

Number	Name	Department	Salary	Location
23603	Jones	413	1100	New Jersey
24568	Kerwin	413	2000	New Jersey
34589	Larson	642	1800	Los Angeles
35761	Myers	611	1400	Orlando
47132	Neumann	413	9000	New Jersey
78321	Stephens	611	8500	Orlando

Row

Primary key

Column

`Employee` table sample data.

Relational-Database Model (Cont.)

Department	Location
413	New Jersey
611	Orlando
642	Los Angeles

Result of selecting distinct Department and Location data from the Employee table.

Relational Database Overview: The books Database

- Sample `books` database
  - Four tables
    - `authors`, `publishers`, `authorISBN`, and `titles`
  - *Foreign key* is table entry that is a primary key in another table (enable rows from multiple tables to be joined)

Relational Database Overview: The books Database (Cont.)

Column	Description
`authorID`	Author's ID number in the database. In the `books` database, this integer column is defined as *autoincremented*. For each row inserted in this table, the database automatically increments the `authorID` value to ensure that each row has a unique `authorID`. This column represents the table's primary key.
`firstName`	Author's first name (a string).
`lastName`	Author's last name (a string).
`authors` table from `books`.	

`authorID`	**`firstName`**	**`lastName`**
1	Harvey	Deitel
2	Paul	Deitel
3	Tem	Nieto
4	Sean	Santry
Sample data from the `authors` table.		

Relational Database Overview: The `books` Database (Cont.)

Column	Description
`publisherID`	The publisher's ID number in the database. This autoincremented integer is the table's primary key.
`publisherName`	The name of the publisher (a string).
`publishers` table from `books`.	

`publisherID`	**`publisherName`**
1	Prentice Hall
2	Prentice Hall PTG
Data from the `publishers` table.	

Relational Database Overview: The books Database (Cont.)

Column	Description
`isbn`	ISBN of the book (a string). The table's primary key.
`title`	Title of the book (a string).
`editionNumber`	Edition number of the book (an integer).
`copyright`	Copyright year of the book (a string).
`publisherID`	Publisher's ID number (an integer). A foreign key to the `publishers` table.
`imageFile`	Name of the file containing the book's cover image (a string).
`price`	Suggested retail price of the book (a real number). [*Note:* The prices shown in this book are for example purposes only.]
`titles` table from `books`.	

Relational Database Overview: The books Database (Cont.)

isbn	title	edition-Number	copy-right	publish-erID	imageFile	price
0130895725	C How to Program	3	2001	1	chtp3.jpg	74.95
0130384747	C++ How to Program	4	2002	1	cpphtp4.jpg	74.95
0130461342	Java Web Services for Experienced Programmers	1	2002	1	jwsfep1.jpg	54.95
0131016210	Java How to Program	5	2003	1	jhtp5.jpg	74.95
0130852473	The Complete Java 2 Training Course	5	2002	2	javactc5.jpg	109.95
0130895601	Advanced Java 2 Platform How to Program	1	2002	1	advjhtp1.jpg	74.95

Sample data from the `titles` table of books.

Relational Database Overview: The `books` Database (Cont.)

Column	Description
`authorID`	The author's ID number, a foreign key to the `authors` table.
`isbn`	The ISBN for a book, a foreign key to the `titles` table..
`authorISBN` table from `books`.	

`authorID`	**`isbn`**	**`authorID`**	**`isbn`**
1	0130895725	2	0139163050
2	0130895725	3	0130829293
2	0132261197	3	0130284173
2	0130895717	3	0130284181
2	0135289106	4	0130895601
Sample data from the `authorISBN` table of `books`.			

Relational Database Overview: The `books` Database (Cont.)

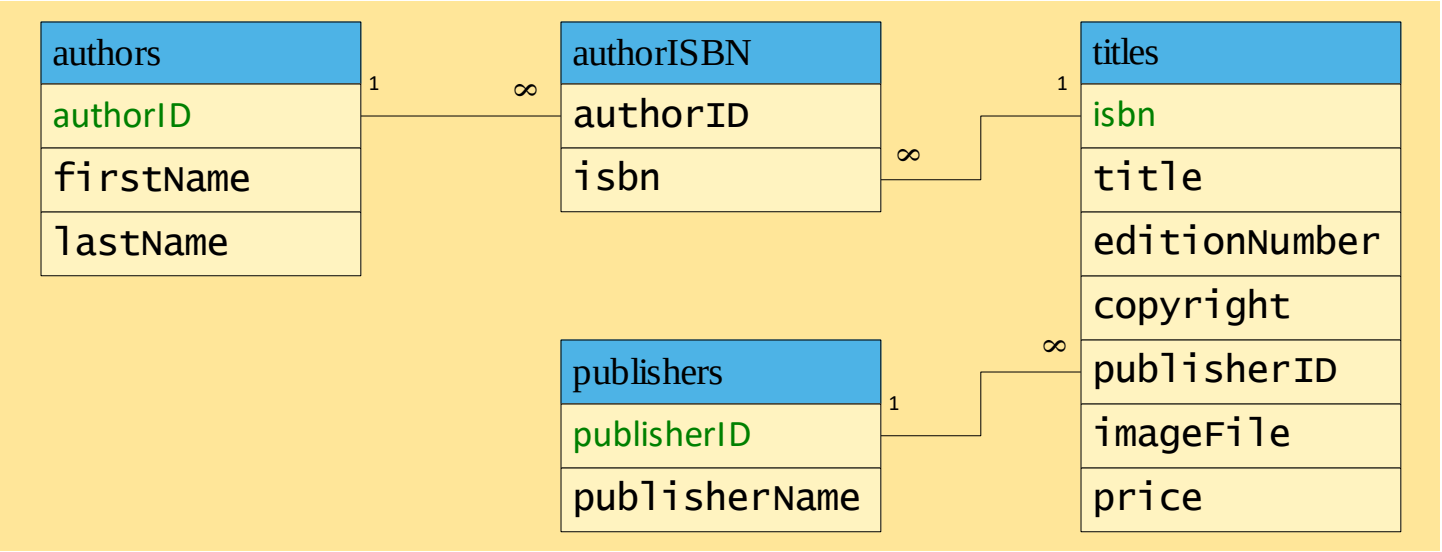


Table relationships in `books`. Entity-relationship (ER) diagram.

SQL

- SQL keywords

SQL keyword	Description
SELECT	Retrieves data from one or more tables.
FROM	Tables involved in the query. Required in every SELECT.
WHERE	Criteria for selection that determine the rows to be retrieved, deleted or updated.
GROUP BY	Criteria for grouping rows.
ORDER BY	Criteria for ordering rows.
INNER JOIN	Merge rows from multiple tables.
INSERT	Insert rows into a specified table.
UPDATE	Update rows in a specified table.
DELETE	Delete rows from a specified table.

SQL query keywords.

Basic SELECT Query

- Simplest form of a SELECT query
  - **SELECT** * **FROM** *tableName*
    - **SELECT** * **FROM** **authors**
    - * means all columns (not recommended)
- Select specific fields from a table
  - **SELECT** **authorID**, **lastName** **FROM** **authors**

authorID	lastName
1	Deitel
2	Deitel
3	Nieto
4	Santry

Sample `authorID` and `lastName` data from the `authors` table.

WHERE Clause

- Specify the selection criteria (predicates)
  - **SELECT** *columnName1, columnName2, …* **FROM** *tableName* **WHERE** *criteria*
    - **SELECT** title, editionNumber, copyright

      FROM titles

      WHERE copyright > 2000

WHERE Clause (Cont.)

title	editionNumber	copyright
C How to Program	3	2001
C++ How to Program	4	2002
The Complete C++ Training Course	4	2002
Internet and World Wide Web How to Program	2	2002
Java How to Program	5	2003
XML How to Program	1	2001
Perl How to Program	1	2001
Advanced Java 2 Platform How to Program	1	2002

Sampling of titles with copyrights after 2000 from table `titles`.

WHERE Clause

- WHERE clause condition operators
  - <, >, <=, >=, =, <>, LIKE
- LIKE (pattern matching)
  - wildcard characters % and _
    - % or * (zero or more characters no matter what they are)
    - _ or ? (single character no matter what it is)
    - wildcard string surrounded by single quotes

WHERE Clause (Cont.)

```
SELECT authorID, firstName, lastName
    FROM authors
    WHERE lastName LIKE 'D%'
```

authorID	firstName	lastName
1	Harvey	Deitel
2	Paul	Deitel
Authors whose last name starts with `D` from the `authors` table.		

WHERE Clause (Cont.)

- **SELECT** authorID, firstName, lastName
  FROM authors
  WHERE lastName **LIKE** '_i%'

authorID	firstName	lastName
3	Tem	Nieto
The only author from the authors table whose last name contains i as the second letter.		

ORDER BY Clause

- Optional **ORDER BY** clause
  - **SELECT** *columnName1, columnName2, …* **FROM** *tableName* **ORDER BY** *column* [**ASC**]
  - **SELECT** *columnName1, columnName2, …* **FROM** *tableName* **ORDER BY** *column* **DESC**
- Note that ASC is default (thus optional)
- **ORDER BY** multiple fields
  - **ORDER BY** *column1 sortingOrder, column2 sortingOrder, …*
- Combine the **WHERE** and **ORDER BY** clauses

ORDER BY Clause (Cont.)

- **SELECT** authorID, firstName, lastName
  FROM authors
  ORDER BY lastName **ASC**

authorID	firstName	lastName
2	Paul	Deitel
1	Harvey	Deitel
3	Tem	Nieto
4	Sean	Santry

Sample data from table `authors` in ascending order by `lastName`.

ORDER BY Clause (Cont.)

- **SELECT** authorID, firstName, lastName
  FROM authors
  ORDER BY lastName **DESC**

authorID	firstName	lastName
4	Sean	Santry
3	Tem	Nieto
2	Paul	Deitel
1	Harvey	Deitel

Sample data from table `authors` in descending order by `lastName`.

ORDER BY Clause (Cont.)

- **SELECT** authorID, firstName, lastName
  FROM authors
  ORDER BY lastName, firstName

authorID	firstName	lastName
1	Harvey	Deitel
2	Paul	Deitel
3	Tem	Nieto
4	Sean	Santry

Sample author data from table `authors` in ascending order by `lastName` and by `firstName`.

ORDER BY Clause (Cont.)

- **SELECT** isbn, title, editionNumber, copyright, price
  FROM titles **WHERE** title **LIKE** '%How to Program'
  ORDER BY title **ASC**

isbn	title	edition-Number	copy-right	price
0130895601	Advanced Java 2 Platform How to Program	1	2002	74.95
0130895725	C How to Program	3	2001	74.95
0130384747	C++ How to Program	4	2002	74.95
0130308978	Internet and World Wide Web How to Program	2	2002	74.95
0130284181	Perl How to Program	1	2001	74.95
0134569555	Visual Basic 6 How to Program	1	1999	74.95
0130284173	XML How to Program	1	2001	74.95
013028419x	e-Business and e-Commerce How to Program	1	2001	74.95

Sampling of books from table `titles` whose titles end with `How to Program` in ascending order by `title`.

Merging Data from Multiple Tables: Joining

- Split related data into separate tables to avoid redundancy
- Join the tables
  - Merge data from multiple tables into a single view
  - INNER JOIN
    - **SELECT** *columnName1, columnName2, …*
      FROM *table1*
      INNER JOIN *table2*
      ON *table1.columnName = table2.column2Name*
    - **SELECT** firstName, lastName, isbn
      FROM authors
      INNER JOIN authorISBN
      ON authors.authorID = authorISBN.authorID
      ORDER BY lastName, firstName

Merging Data from Multiple Tables: Joining (Cont.)

firstName	lastName	isbn	firstName	lastName	isbn
Harvey	Deitel	0130895601	Paul	Deitel	0130895717
Harvey	Deitel	0130284181	Paul	Deitel	0132261197
Harvey	Deitel	0134569555	Paul	Deitel	0130895725
Harvey	Deitel	0139163050	Paul	Deitel	0130829293
Harvey	Deitel	0135289106	Paul	Deitel	0134569555
Harvey	Deitel	0130895717	Paul	Deitel	0130829277
Harvey	Deitel	0130284173	Tem	Nieto	0130161438
Harvey	Deitel	0130829293	Tem	Nieto	013028419x
Paul	Deitel	0130852473	Sean	Santry	0130895601

Sampling of authors and ISBNs for the books they have written in ascending order by `lastName` and `firstName`.

INSERT Statement

- Insert a row into a table
  - **INSERT INTO** *tableName* (*columnName1*, … , *columnNameN*)

    VALUES (*value1*, … , *valueN*)
    - **INSERT INTO** authors (firstName, lastName)

authorID	firstName	lastName
1	Harvey	Deitel
2	Paul	Deitel
3	Tem	Nieto
4	Sean	Santry
5	Sue	Smith

Sample data from table `Authors` after an `INSERT` operation.

INSERT Statement

Insert a row into a table

INSERT INTO *tableName* (*columnName1*, … , *columnNameN*)

VALUES (*value1*, … , *valueN*)

INSERT INTO authors (firstName, lastName)

VALUES ('Sue', 'Smith')

authorID	firstName	lastName
1	Harvey	Deitel
2	Paul	Deitel
3	Tem	Nieto
4	Sean	Santry
5	Sue	Smith

Sample data from table `Authors` after an `INSERT` operation.

INSERT Statement

Insert a row into a table

INSERT INTO tableName (columnName1, … , columnNameN)
VALUES (value1, … , valueN)

INSERT INTO authors (firstName, lastName)
VALUES (‘Sue’, ‘Smith’)

authorID	firstName	lastName
1	Harvey	Deitel
2	Paul	Deitel
3	Tem	Nieto
4	Sean	Santry
5	Sue	Smith

Sample data from table `Authors` after an `INSERT` operation.

UPDATE Statement

- Modify data in a table
  - **UPDATE** *tableName*

    SET *columnName1 = value1, … , columnNameN = valueN*

    WHERE *criteria*

    - **UPDATE** authors

      SET lastName = ‘Jones’

      WHERE lastName = ‘Smith’ **AND** firstName = ‘Sue’

authorID	firstName	lastName
1	Harvey	Deitel
2	Paul	Deitel
3	Tem	Nieto
4	Sean	Santry
5	Sue	Jones

Sample data from table authors after an UPDATE operation.

DELETE Statement

- Remove data from a table (row or rows)
  - **DELETE FROM** *tableName* **WHERE** *criteria*
    - **DELETE FROM** authors

      WHERE lastName = 'Jones' **AND** firstName = 'Sue'

authorID	firstName	lastName
1	Harvey	Deitel
2	Paul	Deitel
3	Tem	Nieto
4	Sean	Santry

Sample data from table `authors` after a `DELETE` operation.

Manipulating Databases with JDBC

- Connect to a database
- Query the database
- Display the results of the query

Connecting to and Querying a Database

DisplayAuthors

- Retrieves the entire **`authors`** table
- Displays the data in a **`JTextArea`**
- **`Connection`** object manages connection between Java program and database

```
connection = DriverManager.getConnection
(DATABASE_URL);
```

- URL `jdbc:db2j:books` specifies communication protocol (jdbc), subprotocol (db2j), name of database (books)
- `getConnection` overloaded (one version can be used to supply account and password)

Connecting to and Querying a Database

```java
// Fig. 23.26: DisplayAuthors.java
// Displaying the contents of the authors table.

import java.awt.*;
import java.sql.*;
import java.util.*;
import javax.swing.*;

public class DisplayAuthors extends JFrame {

   // JDBC driver name and database URL
   static final String JDBC_DRIVER = "com.ibm.db2j.jdbc.DB2jDriver";
   static final String DATABASE_URL = "jdbc:db2j:books";

   // declare Connection and Statement for accessing
   // and querying database
   private Connection connection;
   private Statement statement;

   // constructor connects to database, queries database, processes
   // results and displays results in window
   public DisplayAuthors()
   {
      super( "Authors Table of Books Database" );

```

Imports package `java.sql`, which contains classes and interfaces for the JDBC API.

Example

```
26          // connect to database books and query database
27          try {
28
29             // specify location of database on filesystem
30             System.setProperty( "db2j.system.home", "c:/Cloudscape_5.0" );
31
32             // load database driver class
33             Class.forName( JDBC_DRIVER );
34
35             // establish connection to database
36             connection = DriverManager.getConnection( DATABASE_URL );
37
38             // create Statement for querying database
39             statement = connection.createStatement();
40
41             // query database
42             ResultSet resultSet =
43                statement.executeQuery( "SELECT * FROM authors" );
44
45             // process query results
46             StringBuffer results = new StringBuffer();
47             ResultSetMetaData metaData = resultSet.getMetaData();
48             int numberOfColumns = metaData.getColumnCount();
49
```

Specify location of database

Loads the class definition for the database driver.

Invokes `Connection` method `createStatement` to obtain an object that implements interface `Statement`.

Use the `Statement` object's `executeQuery` method to execute a query that selects all the author information from table `authors`.

Obtains the metadata for the `ResultSet`.

Example

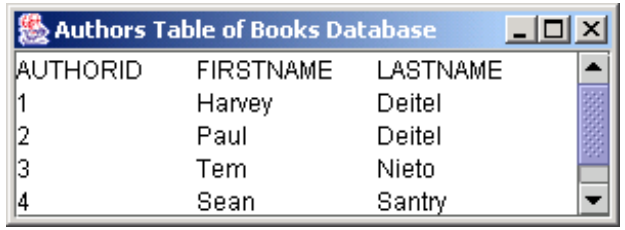
```
50            for ( int i = 1; i <= numberOfColumns; i++ )
51               results.append( metaData.getColumnName( i ) + "\t" );
52
53            results.append( "\n" );
54
55            while ( resultSet.next() ) {
56
57               for ( int i = 1; i <= numberOfColumns; i++ )
58                  results.append( resultSet.getObject( i ) + "\t" );
59
60               results.append( "\n" );
61            }
62
63            // set up GUI and display window
64            JTextArea textArea = new JTextArea( results.toString() );
65            Container container = getContentPane();
66
67            container.add( new JScrollPane( textArea ) );
68
69            setSize( 300, 100 );  // set window size
70            setVisible( true );   // display window
71
72         }  // end try
73
```

Example

```
74          // detect problems interacting with the database
75          catch ( SQLException sqlException ) {
76             JOptionPane.showMessageDialog( null, sqlException.getMessage(),
77                "Database Error", JOptionPane.ERROR_MESSAGE );
78
79             System.exit( 1 );
80          }
81
82          // detect problems loading database driver
83          catch ( ClassNotFoundException classNotFound ) {
84             JOptionPane.showMessageDialog( null, classNotFound.getMessage(),
85                "Driver Not Found", JOptionPane.ERROR_MESSAGE );
86
87             System.exit( 1 );
88          }
89
90          // ensure statement and connection are closed properly
91          finally {
92
93             try {
94                statement.close();
95                connection.close();
96             }
97
```

Example

```
98          // handle exceptions closing statement and connection
99          catch ( SQLException sqlException ) {
100            JOptionPane.showMessageDialog( null,
101               sqlException.getMessage(), "Database Error",
102               JOptionPane.ERROR_MESSAGE );
103
104            System.exit( 1 );
105          }
106       }
107
108    }  // end DisplayAuthors constructor
109
110    // launch the application
111    public static void main( String args[] )
112    {
113       DisplayAuthors window = new DisplayAuthors();
114       window.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
115    }
116
117 }  // end class DisplayAuthors
```



Querying the books Database

- Allow the user to enter any query into the program
- Display the results of a query in a `JTable` (GUI component that looks like a table)
- `ResultSetTableModel` constructor throws any exceptions back to the application that created the `ResultSetTableModel` object

Querying the books Database

```java
// Fig. 23.27: ResultSetTableModel.java
// A TableModel that supplies ResultSet data to a JTable.

import java.sql.*;
import java.util.*;
import javax.swing.table.*;

// ResultSet rows and columns are counted from 1 and JTable
// rows and columns are counted from 0. When processing
// ResultSet rows or columns for use in a JTable, it is
// necessary to add 1 to the row or column number to manipulate
// the appropriate ResultSet column (i.e., JTable column 0 is
// ResultSet column 1 and JTable row 0 is ResultSet row 1).
public class ResultSetTableModel extends AbstractTableModel {
   private Connection connection;
   private Statement statement;
   private ResultSet resultSet;
   private ResultSetMetaData metaData;
   private int numberOfRows;

   // keep track of database connection status
   private boolean connectedToDatabase = false;

```

Querying the books Database

```
24      // initialize resultSet and obtain its meta data object;
25      // determine number of rows
26      public ResultSetTableModel( String driver, String url,
27         String query ) throws SQLException, ClassNotFoundException
28      {
29         // load database driver class
30         Class.forName( driver );
31
32         // connect to database
33         connection = DriverManager.getConnection( url );
34
35         // create Statement to query database
36         statement = connection.createStatement(
37            ResultSet.TYPE_SCROLL_INSENSITIVE,
38            ResultSet.CONCUR_READ_ONLY );
39
40         // update database connection status
41         connectedToDatabase = true;
42
43         // set query and execute it
44         setQuery( query );
45      }
46
```

Querying the books Database

```
47      // get class that represents column type
48      public Class getColumnClass( int column ) throws IllegalStateException
49      {
50         // ensure database connection is available
51         if ( !connectedToDatabase )
52            throw new IllegalStateException( "Not Connected to Database" );
53
54         // determine Java class of column
55         try {
56            String className = metaData.getColumnClassName( column + 1 );
57
58            // return Class object that represents className
59            return Class.forName( className );
60         }
61
62         // catch SQLExceptions and ClassNotFoundExceptions
63         catch ( Exception exception ) {
64            exception.printStackTrace();
65         }
66
67         // if problems occur above, assume type Object
68         return Object.class;
69      }
70
```

Querying the books Database

```
71      // get number of columns in ResultSet
72      public int getColumnCount() throws IllegalStateException
73      {
74         // ensure database connection is available
75         if ( !connectedToDatabase )
76            throw new IllegalStateException( "Not Connected to Database" );
77
78         // determine number of columns
79         try {
80            return metaData.getColumnCount();
81         }
82
83         // catch SQLExceptions and print error message
84         catch ( SQLException sqlException ) {
85            sqlException.printStackTrace();
86         }
87
88         // if problems occur above, return 0 for number of columns
89         return 0;
90      }
91
92      // get name of a particular column in ResultSet
93      public String getColumnName( int column ) throws IllegalStateException
94      {
95         // ensure database connection is available
96         if ( !connectedToDatabase )
97            throw new IllegalStateException( "Not Connected to Database" );
```

Querying the books Database

```
98
99          // determine column name
100         try {
101            return metaData.getColumnName( column + 1 );
102         }
103
104         // catch SQLExceptions and print error message
105         catch ( SQLException sqlException ) {
106            sqlException.printStackTrace();
107         }
108
109         // if problems, return empty string for column name
110         return "";
111      }
112
113      // return number of rows in ResultSet
114      public int getRowCount() throws IllegalStateException
115      {
116         // ensure database connection is available
117         if ( !connectedToDatabase )
118            throw new IllegalStateException( "Not Connected to Database" );
119
120         return numberOfRows;
121      }
122
```

Querying the books Database

```
123     // obtain value in particular row and column
124     public Object getValueAt( int row, int column )
125        throws IllegalStateException
126     {
127        // ensure database connection is available
128        if ( !connectedToDatabase )
129           throw new IllegalStateException( "Not Connected to Database" );
130
131        // obtain a value at specified ResultSet row and column
132        try {
133           resultSet.absolute( row + 1 );
134
135           return resultSet.getObject( column + 1 );
136        }
137
138        // catch SQLExceptions and print error message
139        catch ( SQLException sqlException ) {
140           sqlException.printStackTrace();
141        }
142
143        // if problems, return empty string object
144        return "";
145     }
146
```

Querying the books Database

```
147     // set new database query string
148     public void setQuery( String query )
149        throws SQLException, IllegalStateException
150     {
151        // ensure database connection is available
152        if ( !connectedToDatabase )
153           throw new IllegalStateException( "Not Connected to Database" );
154
155        // specify query and execute it
156        resultSet = statement.executeQuery( query );
157
158        // obtain meta data for ResultSet
159        metaData = resultSet.getMetaData();
160
161        // determine number of rows in ResultSet
162        resultSet.last();                    // move to last row
163        numberOfRows = resultSet.getRow();   // get row number
164
165        // notify JTable that model has changed
166        fireTableStructureChanged();
167     }
168
```

Querying the books Database

```
169     // close Statement and Connection
170     public void disconnectFromDatabase()
171     {
172        // close Statement and Connection
173        try {
174           statement.close();
175           connection.close();
176        }
177
178        // catch SQLExceptions and print error message
179        catch ( SQLException sqlException ) {
180           sqlException.printStackTrace();
181        }
182
183        // update database connection status
184        finally {
185           connectedToDatabase = false;
186        }
187     }
188
189  }  // end class ResultSetTableModel
```

Querying the books Database

```
169     // close Statement and Connection
170     public void disconnectFromDatabase()
171     {
172        // close Statement and Connection
173        try {
174           statement.close();
175           connection.close();
176        }
177
178        // catch SQLExceptions and print error message
179        catch ( SQLException sqlException ) {
180           sqlException.printStackTrace();
181        }
182
183        // update database connection status
184        finally {
185           connectedToDatabase = false;
186        }
187     }
188
189  }  // end class ResultSetTableModel
```

Querying the books Database

- Resultset

`ResultSet static` concurrency constant	Description
`CONCUR_READ_ONLY`	Specifies that a `ResultSet` cannot be updated (i.e., changes to the `ResultSet` contents cannot be reflected in the database with `ResultSet`'s *update* methods).
`CONCUR_UPDATABLE`	Specifies that a `ResultSet` can be updated (i.e., changes to the `ResultSet` contents can be reflected in the database with `ResultSet`'s *update* methods).
`ResultSet` constants for specifying result properties.	

Querying the books Database

```
// Fig. 23.30: DisplayQueryResults.java
// Display the contents of the Authors table in the
// Books database.

import java.awt.*;
import java.awt.event.*;
import java.sql.*;
import java.util.*;
import javax.swing.*;
import javax.swing.table.*;

public class DisplayQueryResults extends JFrame {

   // JDBC driver and database URL
   static final String JDBC_DRIVER = "com.ibm.db2j.jdbc.DB2jDriver";
   static final String DATABASE_URL = "jdbc:db2j:books";

   // default query selects all rows from authors table
   static final String DEFAULT_QUERY = "SELECT * FROM authors";

   private ResultSetTableModel tableModel;
   private JTextArea queryArea;

     // create ResultSetTableModel and GUI
     public DisplayQueryResults()
     {
        super( "Displaying Query Results" );
```

Querying the books Database

```
28
29          // create ResultSetTableModel and display database table
30          try {
31
32             // specify location of database on filesystem
33             System.setProperty( "db2j.system.home", "C:/Cloudscape_5.0" );
34
35             // create TableModel for results of query SELECT * FROM authors
36             tableModel = new ResultSetTableModel( JDBC_DRIVER, DATABASE_URL,
37                DEFAULT_QUERY );
38
39             // set up JTextArea in which user types queries
40             queryArea = new JTextArea( DEFAULT_QUERY, 3, 100 );
41             queryArea.setWrapStyleWord( true );
42             queryArea.setLineWrap( true );
43
44             JScrollPane scrollPane = new JScrollPane( queryArea,
45                ScrollPaneConstants.VERTICAL_SCROLLBAR_AS_NEEDED,
46                ScrollPaneConstants.HORIZONTAL_SCROLLBAR_NEVER );
47
48             // set up JButton for submitting queries
49             JButton submitButton = new JButton( "Submit Query" );
50
51             // create Box to manage placement of queryArea and
52             // submitButton in GUI
53             Box box = Box.createHorizontalBox();
54             box.add( scrollPane );
55             box.add( submitButton );
56
```

Querying the books Database

```
57          // create JTable delegate for tableModel
58          JTable resultTable = new JTable( tableModel );
59
60          // place GUI components on content pane
61          Container c = getContentPane();
62          c.add( box, BorderLayout.NORTH );
63          c.add( new JScrollPane( resultTable ), BorderLayout.CENTER );
64
65          // create event listener for submitButton
66          submitButton.addActionListener(
67
68             new ActionListener() {
69
70                // pass query to table model
71                public void actionPerformed( ActionEvent event )
72                {
73                   // perform a new query
74                   try {
75                      tableModel.setQuery( queryArea.getText() );
76                   }
77
78                      // catch SQLExceptions when performing a new query
79                      catch ( SQLException sqlException ) {
80                         JOptionPane.showMessageDialog( null,
81                            sqlException.getMessage(), "Database error",
82                            JOptionPane.ERROR_MESSAGE );
83
```

Querying the books Database

```
84                          // try to recover from invalid user query
85                          // by executing default query
86                          try {
87                             tableModel.setQuery( DEFAULT_QUERY );
88                             queryArea.setText( DEFAULT_QUERY );
89                          }
90
91                          // catch SQLException when performing default query
92                          catch ( SQLException sqlException2 ) {
93                             JOptionPane.showMessageDialog( null,
94                                sqlException2.getMessage(), "Database error",
95                                JOptionPane.ERROR_MESSAGE );
96
97                             // ensure database connection is closed
98                             tableModel.disconnectFromDatabase();
99
100                           System.exit( 1 );    // terminate application
101
102                        }  // end inner catch
103
104                     } // end outer catch
105
106                  }  // end actionPerformed
107
108               }  // end ActionListener inner class
109
110            ); // end call to addActionListener
111
```

Querying the books Database

```
112            // set window size and display window
113            setSize( 500, 250 );
114            setVisible( true );
115
116         }  // end try
117
118         // catch ClassNotFoundException thrown by
119         // ResultSetTableModel if database driver not found
120         catch ( ClassNotFoundException classNotFound ) {
121            JOptionPane.showMessageDialog( null,
122               "Cloudscape driver not found", "Driver not found",
123               JOptionPane.ERROR_MESSAGE );
124
125            System.exit( 1 );   // terminate application
126         } // end catch
127
128         // catch SQLException thrown by ResultSetTableModel
129         // if problems occur while setting up database
130         // connection and querying database
131         catch ( SQLException sqlException ) {
132            JOptionPane.showMessageDialog( null, sqlException.getMessage(),
133               "Database error", JOptionPane.ERROR_MESSAGE );
134
135            // ensure database connection is closed
136            tableModel.disconnectFromDatabase();
137
138            System.exit( 1 );   // terminate application
139         }
140
```

Querying the books Database

```
141         // dispose of window when user quits application (this overrides
142         // the default of HIDE_ON_CLOSE)
143         setDefaultCloseOperation( DISPOSE_ON_CLOSE );
144
145         // ensure database connection is closed when user quits application
146         addWindowListener(
147
148            new WindowAdapter() {
149
150               // disconnect from database and exit when window has closed
151               public void windowClosed( WindowEvent event )
152               {
153                  tableModel.disconnectFromDatabase();
154                  System.exit( 0 );
155               }
156            }
157         );
158
159      }  // end DisplayQueryResults constructor
160
161      // execute application
162      public static void main( String args[] )
163      {
164         new DisplayQueryResults();
165      }
166
167   }  // end class DisplayQueryResults
```

Querying the books Database

Displaying Query Results

SELECT * FROM authors

Submit Query

AUTHORID	FIRSTNAME	LASTNAME
1	Harvey	Deitel
2	Paul	Deitel
3	Tem	Nieto
4	Sean	Santry

Displaying Query Results

SELECT lastName, firstName, title, editionNumber FROM authors, titles, authorISBN WHERE authors.authorID = authorISBN.authorID AND titles.isbn = authorISBN.isbn

Submit Query

LASTNAME	FIRSTNAME	TITLE	EDITIONNUMBER
Deitel	Harvey	C How to Program	3
Deitel	Harvey	C How to Program	2
Deitel	Harvey	C++ How to Program	3
Deitel	Harvey	C++ How to Program	2
Deitel	Harvey	The Complete C++ ...	3
Deitel	Harvey	e-Business and e-...	1
Deitel	Harvey	Internet and World ...	1
Deitel	Harvey	The Complete Inter...	1
Deitel	Harvey	Java How to Progra...	3
Deitel	Harvey	Java How to Progra...	2

Java to SQL Server Database Connectivity

Using JDBC

JDBC

- JDBC is an interface between Java and Database
- JDBC receives queries from Java Application program and communicate with Database
- All the communications are in the form of SQL commands
  - JDBC is responsible for
  - Open a Connection
  - Communicate with database
  - Execute SQL statements
  - Retrieve query results

Open Database Connectivity

- Standard designed by Microsoft to interact with databases
- ODBC is packed with many features and extending support for all type of databases
- ODBC provides multiple mechanism for performing single task and number of data handling capabilities

JDBC vs ODBC

- JDBC and ODBC are X/OPEN call level interface for SQL JDBC is not a derivative of ODBC
- JDBC is compact and simple
- JDBC is meant for simple access to the database and difficult task at least make possible

JDBC Drivers

- JDBC driver is responsible for making connection with different databases
- It is also translating the queries received from Application and submit into database
- A reverse translation is also required to perform by the Driver
- JDBC Driver speaks JAVA to Application and native language to database
- JDBC Drivers are exists for almost all databases
- Appropriate driver will load for required database

JDBC – ODBC Bridge

- It is a JDBC driver designed to let Java application communicate with database via an underlying ODBC driver
- It is called Type I JDBC connector
- It can be used with multiple databases and is vendor independent
- This type JDBC driver speaks only to ODBC driver, hence works for Databases supported by ODBC
- One more added layer is used and hence more complex and slower than JDBC drivers

Native-API-Partly-Java Driver

- It make use of local native libraries to communicate with database
- Vendor specific Call Level Interface(CLI) installed locally are used by this type driver
- CLI libraries are actually communicate with database
- Application level requests are translated into equallent native method call of CLI
- Faster than Type I driver

JDBC-Net-All-Java Driver

- Type III Driver, uses the CLI libraries locates in a remote server
- Type III driver has two major components
  - An All-Java portion that can download to the client
  - Server portion containing both Java and Native methods
- All communication between Application and Database is 100% Java to Java
- This type of driver is also depending on CLI calls, which is installed on Server
- Type III can be used in Internet, since no direct access to CLI libraries
- Type III Network protocol is not standardized

Native-Protocol-All-Java Driver

- 100% java specific drivers
- No intermediate translation is required
- But all vendor specific driver cannot released by Java
- Java Applets are now free from Access restrictions

JDBC Implementation

Seven Steps

– Import java.sql package

– Load and register the driver

– Establish a connection to the database server

– Create a statement

– Execute the statement

– Retrieve the result

– Close the statement and connection

Load and Register Driver

Class.forName("Driver ClassName");

Eg:1 Class.forName(“sun.jdbc.odbc.JdbcOdbcDriver”)

2 Class.forName("com.mysql.jdbc.Driver");

Note: Calling the Class.forName automatically creates an instance of a driver and registers it with the DriverManager, so you don't need to create an instance of the class

Establish Connection

Connection conn=DriverManager.getConnection("URL");

The drivers loaded recognizes, the JDBC URL in DriverManager.getConnection, that driver establishes a connection to the DBMS specified in the JDBC URL.

The DriverManager class,manages all of the details of establishing the connection

The connection returned by the method DriverManager.getConnection is an open connection you can use to create JDBC statements that pass your SQL statements to the DBMS.

DriverManager.getConnection(“jdbc:mysql://172.16.5.27/campusdb”,”mca”,”mca”);

Managing Statement

createStatement() of Connection class is used to make object of Statements.

Eg: Statement stat=con.createStatement();

Statement object can call executeQuery("SQL Command") to execute a select statement.

Use executeUpdate("SQL Command") to execute any data updation commands

Resultset

An executeQuery() method retrieves the selected records as an object of ResultSet class

It stores data in tabular format. Rowid and ColID can be used to identify each data.

Rows are records of table and columns are fields of table

A cursor is attached to fetch data from any row

Metadata

Information that describes the structure and properties of your data

Two types of Metadata

- **Resultset Metadata:** Information about the data contained

  in a Resultset, such as column name, number of columns and column data types

- **Database Metadata**: Information about database, such as supported functions, username,

  current transaction isolation level

Any Question

THANK YOU ☺

QC360
Be Compliant