7. Databases and JDBC

2. JDBC Database Access

JDBC Basics

- The JDBC API is a Java API that can access any kind of tabular data, especially data stored in a Relational Database
- JDBC Product Components
  - The JDBC API
  - JDBC Driver Manager
  - JDBC Test Suite
  - JDBC-ODBC Bridge

Eclipse & Derby Projects

- Eclipse: New -> Java Project
- Fill project name and click next
- Click “Add External JARs” button in the libraries tab
- Find derby.jar (usually in Program Files \ Java\jdk1.7.0_xx\db\lib folder) and click Open button
- Click Finish button

Processing SQL Statements with JDBC

1. Establishing a connection
2. Create a statement
3. Execute the query
4. Process the ResultSet object
5. Close the connection

Basic Example I

```
package app;
import java.sql.*;
public class E721JDBCBasics {
    public static void main(String[] args) {
     try{
          // jdbc statements body (see next slide)
     }
     catch(SQLException ex){
          System.out.println("Error " + ex.getMessage());
     }
    }
}
```

Basic Example II

```
Connection con = DriverManager.getConnection
    ("jdbc:derby:C:\\VMO\\Курсы\\Projects\\CM");
Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery(
     "SELECT name, charge FROM merchant");
while (rs.next()){
    String nm = rs.getString("name");
    double p = rs.getDouble(2);
    System.out.println(nm + "   " + p);
}
con.close();
```

See 721JDBCBasics project for the full text

Processing SQL Statements with JDBC

1. Establishing a connection
2. Create a statement
3. Execute the query
4. Process the ResultSet object
5. Close the connection

Establishing a connection.

- A JDBC application connects to a target data source using one of two classes:
  - DriverManager - connects an application to a data source, specified by a database URL
  - DataSource - allows details about the underlying data source to be transparent to your application

Connection example

```
public static Connection getConnection() throws IOException,
                                   SQLException{
    Connection conn = null;
    Properties props = new Properties();
    InputStreamReader in = new InputStreamReader(new
    FileInputStream("appProperties.txt"), "UTF-8");
    props.load(in);
    in.close();

    String connString = props.getProperty("DBConnectionString");
    conn = DriverManager.getConnection(connString);
    return conn;
}
```

See 722JDBCConnection project for the full text

Processing SQL Statements with JDBC

1. Establishing a connection
2. Create a statement
3. Execute the query
4. Process the ResultSet object
5. Close the connection

Creating Statements

- Kinds of statements:
  - Statement - simple SQL statements with no parameters
  - PreparedStatement (extends Statement) - precompiling SQL statements that might contain input parameters
  - CallableStatement (extends PreparedStatement) - used to execute stored procedures that may contain both input and output parameters

Insert New Customer Example I

```
Connection con = getConnection();
String sql = "INSERT INTO customer (name, address, ";
sql += " email, ccNo, ccType, maturity) values(";
sql += " 'Clar Nelis', 'Vosselaar st. 19, Trnaut, Belgium', ";
sql += " 'Clar@adw.com',   '11345694671231', ";
sql += " 'MasterCard', '2014-07-31') ";
Statement stmt = con.createStatement();
stmt.executeUpdate(sql);
con.close();
```

See 723SimpleInsert project for the full text

Prepared Statements

- Usually reduces execution time (the DBMS can just run the PreparedStatement SQL statement without having to compile it first)
- Used most often for SQL statements that take parameters. You can use the same statement and supply it with different values each time you execute it

Insert New Customer Example II

```
public void addCustomer(String name, String address, String email, String
    ccNo, String ccType, java.sql.Date dt) throws SQLException, IOException{
    Connection con = getConnection();
    String sql = "INSERT INTO customer (name, address, ";
    sql += " email, ccNo, ccType, maturity) values(?,?,?,?,?,?) ";
    PreparedStatement stmt = con.prepareStatement(sql);
    stmt.setString(1, name);
    stmt.setString(2, address);
    stmt.setString(3, email);
    stmt.setString(4, ccNo);
    stmt.setString(5, ccType);
    stmt.setDate(6, dt);
    stmt.executeUpdate();
    con.close();
}
```

See 724PreparedInsert project for the full text

SQL Date

From GregorianCalendar:

```
GregorianCalendar c = new GregorianCalendar(2012, 03, 31);
java.util.Date dt = c.getTime();
java.sql.Date dt1 = new java.sql.Date(dt.getTime());
```

From LocalDate:

```
LocalDate dt1 = LocalDate.of(2015, 2, 15);
Instant instant =
    dt1.atStartOfDay(ZoneId.systemDefault()).toInstant();
java.sql.Date dt = new
    java.sql.Date(java.util.Date.from(instant).getTime());
```

Processing SQL Statements with JDBC

1. Establishing a connection
2. Create a statement
3. Execute the query
4. Process the ResultSet object
5. Close the connection

Executing Queries

- executeQuery: Returns one ResultSet object
- executeUpdate: Returns an integer representing the number of rows affected by the SQL statement
- execute: Returns true if the first object that the query returns is a ResultSet object

Exercise: Get Merchant’s Total

- Show total for a merchant which id is given in the first command string parameter.

Exercise: Get Merchant’s Total

- See 725Query project for the full text.

Processing SQL Statements with JDBC

1. Establishing a connection
2. Create a statement
3. Execute the query
4. Process the ResultSet object
5. Close the connection

Processing ResultSet Objects

- You access the data in a ResultSet object through a cursor
- Note that this cursor is not a database cursor
- This cursor is a pointer that points to one row of data in the ResultSet object
- Initially, the cursor is positioned before the first row
- You call various methods defined in the ResultSet object to move the cursor

Exercise: List of Merchants

- Create an application to display list of merchants:
  - Create a Merchant class with fields necessary for saving merchant’s data and getStringForPrint method for displaying these data
  - Create getMerchants method for filling list of merchants from a corresponding data table
  - Process this list of merchants to display it on the system console

Exercise: List of Merchants

- See 726MerchList project for the full text.

Processing SQL Statements with JDBC

1. Establishing a connection
2. Create a statement
3. Execute the query
4. Process the ResultSet object
5. Close the connection

Closing Connections

- Call the method Statement.close to immediately release the resources it is using.
- When you call this method, its ResultSet objects are closed
- finally {

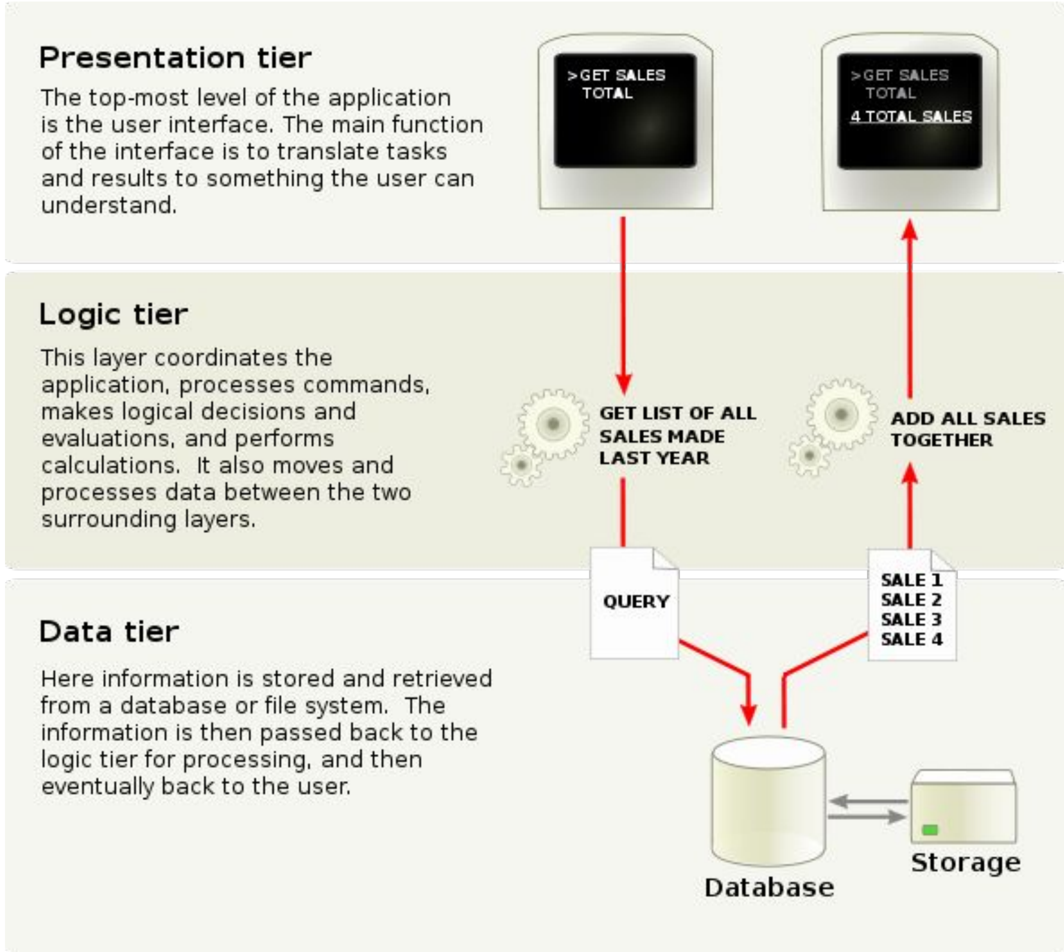
```
    if (stmt != null) { stmt.close(); }
}
```

Closing Connections in Java 7

- Use a try-with-resources statement to automatically close Connection, Statement, and ResultSet objects
- try (Statement stmt = con.createStatement())

```
{
 // ...
}
```

Three-tired application



Handling SQLExceptions

- The SQLException contains the following information
  - A description of the error - getMessage()
  - A SQLState standard code – getSQLState()
  - An error code (DB specific) – getErrorCode()
  - A cause (Throwable objects that caused the SQLException instance to be thrown) – getCause()
  - A reference to any *chained* exceptions – getNextException()

Data Tier

- Separation of concerns principle:
  - business and presentation tiers should not know anything about database structure
  - SQLexceptions should be processed within data tier

Exercise: Add Payment

- Create a method to add new payment info to the database

Exercise: Add Payment

- See 727AddPayment project for the full text.

Transactions

- These statements should take effect only together:

// Insert new record into PAYMENT table

// Update corresponding record in MERCHANT table

- The way to be sure that either both actions occur or neither action occurs is to use a transaction

Using Transactions

```
public static void addPayment(Connection conn,
    java.util.Date dt, int customerId, int merchantId, String
    goods, double total) throws SQLException{
    conn.setAutoCommit(false);
    double charge = getCharge(conn, merchantId);
    if (charge < 0.0) return;
    // Insert new record into PAYMENT table
    // Update corresponding record in MERCHANT table
    conn.commit();
}
```

Rollback Method

- Calling the method rollback terminates a transaction and returns any values that were modified to their previous values.
- If you are trying to execute one or more statements in a transaction and get a SQLException, call the method rollback to end the transaction and start the transaction all over again.

Exercise: Get Income Report

- Create a report about CM system’s income got from each merchant.

Exercise: Get Income Report

- See 728MerchantCharge project for the full text.

Object-Relational Mapping

- SQL DBMS can only store and manipulate scalar values such as integers and strings organized within tables
- Data management tasks in object-oriented programming are typically implemented by manipulating objects that are almost always non-scalar values
- The problem is translating the logical representation of the objects into an atomized form that is capable of being stored on the database

ORM Advantages&Disadvantages

- Advantage:
  - often reduces the amount of code that needs to be written
- Disadvantage:
  - performance problem

Some Java ORM Systems

- Hibernate, open source ORM framework, widely used
- MyBatis, formerly named iBATIS, has .NET port
- Cayenne, Apache, open source for Java
- Athena Framework, open source Java ORM
- CarbonadoCarbonado, open source framework, backed by Berkeley DBCarbonado, open source framework, backed by Berkeley DB or JDBC
- EclipseLink, Eclipse persistence platform
- TopLink by Oracle
- QuickDB ORM, open source ORM framework (GNU LGPL)

Manuals

- http://docs.oracle.com/javase/tutorial/jdbc/index.html