Principles and Techniques of DBMS 4
DB Access II

Haopeng Chen

***RE**liable, **IN**telligent and **S**calable Systems Group (**REINS**)*
Shanghai Jiao Tong University
Shanghai, China
http://reins.se.sjtu.edu.cn/~chenhp
e-mail: chen-hp@sjtu.edu.cn

Overview

- Access Database via JDBC Reading
  - RowSet vs. ResultSet
- Basics of ORM
- Pros and Cons of
  - JDBC/ODBC Reading and ORM

Manipulate result

- ResultSet
  - Types
    1. TYPE_FORWARD_ONLY
    2. TYPE_SCROLL_INSENSITIVE
    3. TYPE_SCROLL_SENSITIVE
  - Concurrency
    1. CONCUR_READ_ONLY
    2. CONCUR_UPDATABLE
  - Holdability
    1. HOLD_CURSORS_OVER_COMMIT
    2. CLOSE_CURSORS_AT_COMMIT

Manipulate result

REliable, INtelligent & Scalable Systems

- ResultSet

```
Statement stmt = conn.createStatement();
ResultSet rs = stmt.executeQuery("select author, title, isbn"
                               +"from booklist");

next()              beforeFirst()
previous()          afterLast()
first()             relative(int rows)
last()              absolute(int row)

int colIdx = rs.findColumn("ISBN");

ResultSetMetaData rsmd = rs.getMetaData();
int colType [] = new int[rsmd.getColumnCount()];
for (int idx = 0, int col = 1; idx < colType.length;
     idx++, col++)
    colType[idx] = rsmd.getColumnType(col);
```

Manipulate result

REliable, INtelligent & Scalable Systems

- ResultSet

```
// Update a row: two-phase process
Statement stmt =
conn.createStatement(ResultSet.TYPE_FORWARD_ONLY,
                ResultSet.CONCUR_UPDATABLE);
ResultSet rs = stmt.executeQuery(“select author from
                booklist " + "where isbn = 140185852”);
rs.next();
rs.updateString(“author”, “Zamyatin, Evgenii Ivanovich”);
rs.updateRow();

// Delete a row
rs.absolute(4);
rs.deleteRow();
```

Manipulate result

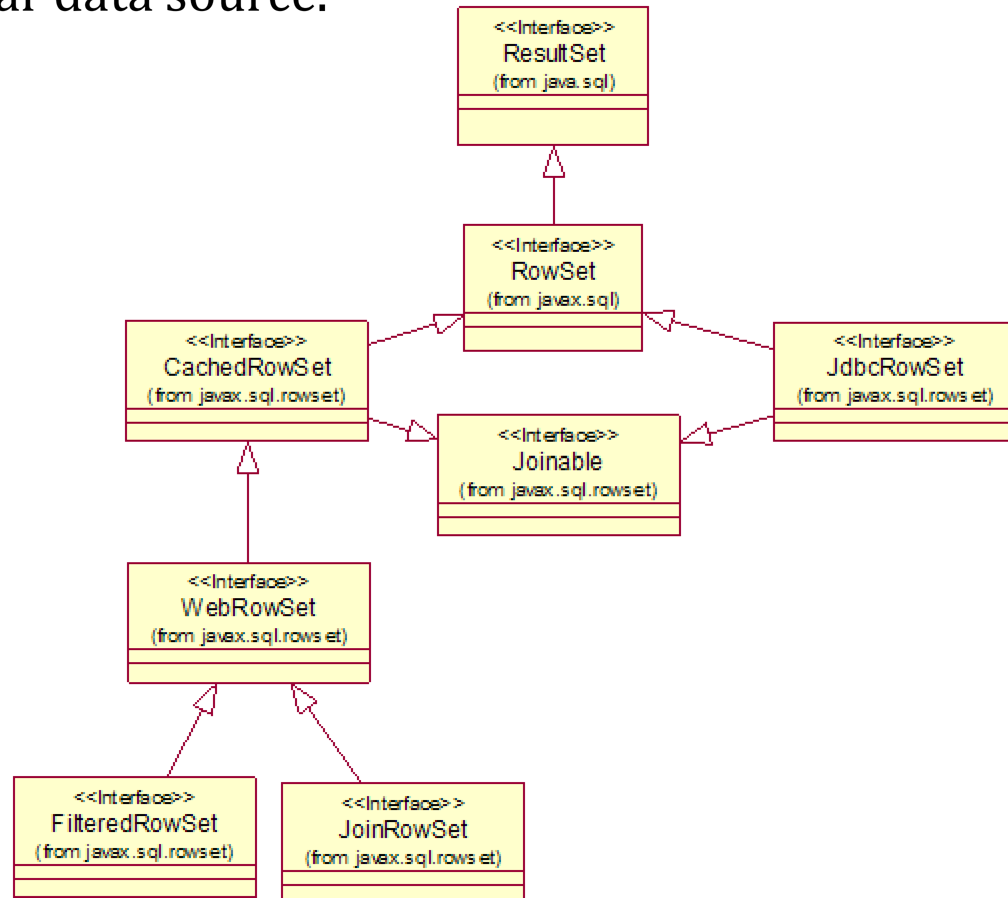
- ResultSet

```
// Insert a row: three steps
// select all the columns from the table booklist
ResultSet rs = stmt.executeQuery(“select author, title,
                                 isbn " + "from booklist”);
rs.moveToInsertRow();
// set values for each column
rs.updateString(1, “Huxley, Aldous”);
rs.updateString(2, “Doors of Perception and Heaven and
                Hell”);
rs.updateLong(3, 60900075);
// insert the row
rs.insertRow();
// move the cursor back to its position in the result set
rs.moveToCurrentRow();
```

Manipulate result

REliable, INtelligent & Scalable Systems

- RowSet
  - A `javax.sql.RowSet` object encapsulates a set of rows that have been retrieved from a tabular data source.
  - JdbcRowSet - online
  - CachedRowSet
    - WebRowSet
    - FilteredRowSet
    - JoinRowSet



Manipulate result

REliable, INtelligent & Scalable Systems

- RowSet
  - JdbcRowSet – online

```
public class AccessDB {
   private JdbcRowSet rowset;
   public AccessDB(String url, String user, String pwd)
                        throws SQLException {
          RowSetFactory rowSetFactory =
                 RowSetProvider.newFactory();
          JdbcRowSet rowset =
                 rowSetFactory.createJdbcRowSet();
          rowset.setUrl(
                 "jdbc:mysql://localhost:3306/sample_one");
          rowset.setUsername("root");
          rowset.setPassword(“12345678");
          rowset.setCommand("SELECT * FROM tbl_user");
          rowset.execute();
   }
```

Manipulate result

REliable, INtelligent & Scalable Systems

- RowSet
  - JdbcRowSet – online

```
public List<User> get() throws SQLException {
    List<User> records = new ArrayList<>();
    rowset.beforeFirst();
    while (rowset.next()) {
        User record = new User();
        record.setId(rowset.getLong(1));
        record.setUsername(rowset.getString(2));
        record.setPassword(rowset.getString(3));
        record.setEmail(rowset.getString(4));
        records.add(record);
    }
    return records;
}
```

Manipulate result

- RowSet
  - JdbcRowSet – online

```
public void add(User user) throws SQLException {
    rowset.moveToInsertRow();
    rowset.updateInt(1, (int)user.getId());
    rowset.updateString(2, user.getUsername());
    rowset.updateString(3, user.getPassword());
    rowset.updateString(4, user.getEmail());
    rowset.insertRow();
}
```

Manipulate result

REliable, INtelligent & Scalable Systems

- RowSet
  - JdbcRowSet – online

```
public class User {

    private long id;
    private String username;
    private String password;
    private String email;

    public long getId(){return this.id;}
    public String getUsername(){return this.username;}
    public String getPassword(){return this.password;}
    public String getEmail(){return this.email;}

    public void setId(long id) {this.id = id;}
    public void setUsername(String username) {this.username = username;}
    public void setPassword(String password) {this.password = password;}
    public void setEmail(String email) {this.email = email;}
}
```

Manipulate result

REliable, INtelligent & Scalable Systems

- RowSet
  - JdbcRowSet – online

```
public static void main(String[] args) throws SQLException {
          AccessDB ad = new AccessDB(url,user,pwd);

          List<User> records = ad.get();
          Iterator<User> it = records.iterator();
          while(it.hasNext()){
                    User use = it.next();
                    ……
          }

          User usertoadd = new User();
          usertoadd.setId(10);
          ……
          ad.add(usertoadd);

          records = ad.get();
          it = records.iterator();
          while(it.hasNext()){
                    User use = it.next();
                    ……
          }
}
```

Manipulate result

REliable, INtelligent & Scalable Systems

- RowSet
  - CachedRowSet

```
RowSetFactory rowSetFactory =
        RowSetProvider.newFactory();
rowset = rowSetFactory.createCachedRowSet();
```

  - WebRowSet

```
RowSetFactory rowSetFactory =
        RowSetProvider.newFactory();
rowset = rowSetFactory.createWebRowSet();
rowset.execute();
rowset.writeXml(System.out);
```

Manipulate result

REliable, INtelligent & Scalable Systems

- RowSet
  - FilteredRowSet

```
RowSetFactory rowSetFactory =
             RowSetProvider.newFactory();
rowset =  rowSetFactory.createFilteredRowSet();
Range range = new Range();
rowset.setFilter(range);
```

Manipulate result

REliable, INtelligent & Scalable Systems

- RowSet
  - FilteredRowSet

```
class Range implements Predicate {

    public boolean evaluate(RowSet rs) {
        try {
            if (rs.getInt(1) > 1) {
                return true;
            }
        } catch (SQLException e) {
            // do nothing
        }
        return false;
    }

    public boolean evaluate(Object value, int column) throws SQLException {
        return false;
    }

    public boolean evaluate(Object value, String columnName)
            throws SQLException {
        return false;
    }
}
```

Manipulate result

REliable, INtelligent & Scalable Systems

- RowSet
  - Rowsets can generate three different types of events:

    1. Cursor movement events
    2. Row change events
    3. Rowset change events

  - To add a listener to Rowset

```
Listener listener = new Listener();
rowset.addRowSetListener(listener);
```

Manipulate result

REliable, INtelligent & Scalable Systems

- RowSet
  - RowSetListener:

```
public class Listener implements RowSetListener {

    @Override
    public void cursorMoved(RowSetEvent arg0) {
        System.out.println("The cursor is moved");
    }

    @Override
    public void rowChanged(RowSetEvent arg0) {
        System.out.println("A row is changed");
    }

    @Override
    public void rowSetChanged(RowSetEvent arg0) {
        System.out.println("The rowset is changed");
    }

}
```

ODBC Specification

- ODBC is a specification for a database API.
  - This API is independent of any one DBMS or operating system; although this manual uses C, the ODBC API is language-independent.
- The functions in the ODBC API are implemented by developers of DBMS-specific drivers.
  - Applications call the functions in these drivers to access data in a DBMS-independent manner.
  - A Driver Manager manages communication between applications and drivers.
- It is important to understand that ODBC is designed to expose database capabilities, not supplement them.

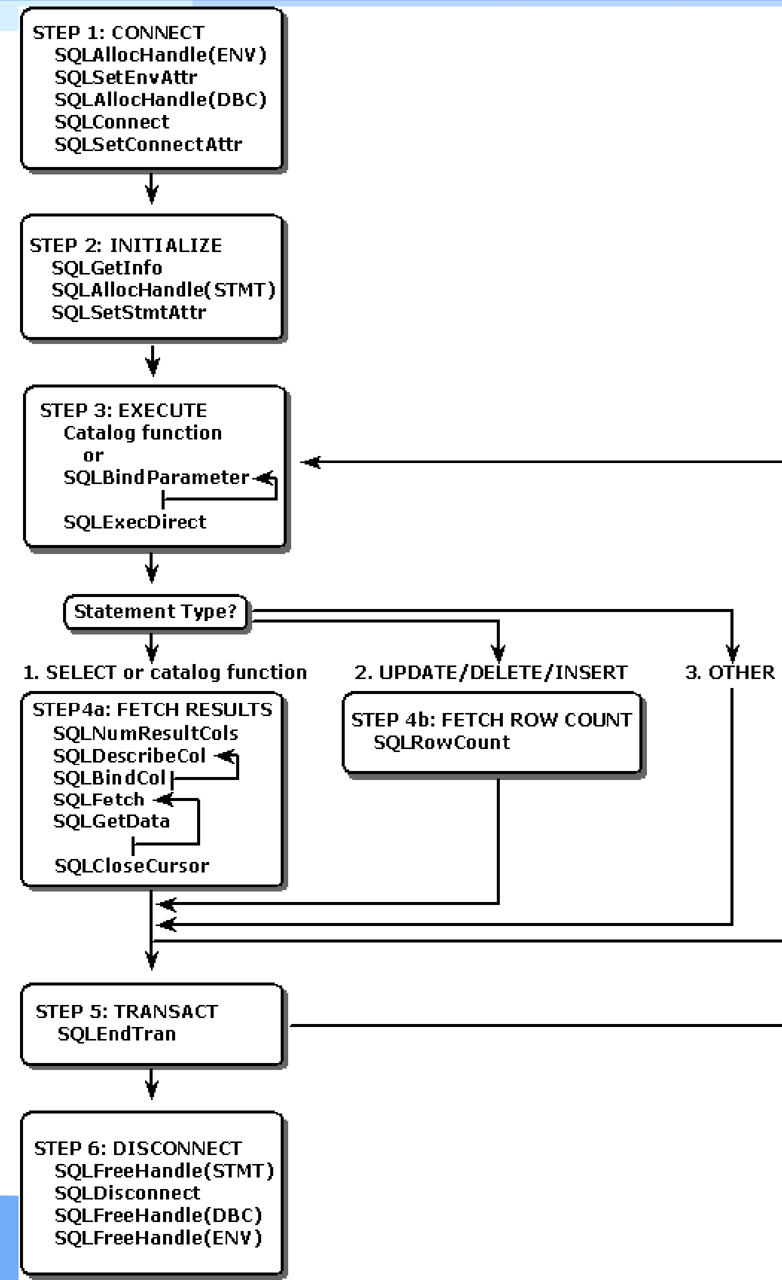
ODBC Architecture

- The ODBC architecture has four components:
  - **Application** Performs processing and calls ODBC functions to submit SQL statements and retrieve results.
  - **Driver Manager** Loads and unloads drivers on behalf of an application. Processes ODBC function calls or passes them to a driver.
  - **Driver** Processes ODBC function calls, submits SQL requests to a specific data source, and returns results to the application. If necessary, the driver modifies an application's request so that the request conforms to syntax supported by the associated DBMS.
  - **Data Source** Consists of the data the user wants to access and its associated operating system, DBMS, and network platform (if any) used to access the DBMS.

Basic ODBC Application Steps

REliable, INtelligent & Scalable Systems

- Step 1: Connect to the Data Source
- Step 2: Initialize the Application
- Step 3: Build and Execute an SQL Statement
- Step 4a: Fetch the Results
- Step 4b: Fetch the Row Count
- Step 5: Commit the Transaction
- Step 6: Disconnect from the Data Source



Access DB with ODBC

REin
REliable, INtelligent & Scalable Systems

```
string MyConString = "DRIVER={MySQL ODBC 3.51 Driver};" +
                              "SERVER=localhost;" + "DATABASE=inv;" +
                              "UID=root;" + "PASSWORD=831025;" + "OPTION=3";
OdbcConnection MyConnection = new OdbcConnection(MyConString);
MyConnection.Open();

string query = "insert into test values( 'hello', 'lucas', 'liu')";
OdbcCommand cmd = new OdbcCommand(query, MyConnection);

try{
  cmd.ExecuteNonQuery();
}
catch(Exception ex){
  Console.WriteLine("record duplicate.");
}finally{
  cmd.Dispose();
}
```

Access DB with ODBC

REliable, INtelligent & Scalable Systems

```
string tmp1 = null;
string tmp2 = null;
string tmp3 = null;
query = "select * from test ";
OdbcCommand cmd2 = new OdbcCommand(query, MyConnection);
OdbcDataReader reader = cmd2.ExecuteReader();

while (reader.Read())
{
      tmp1 = reader[0].ToString();
      tmp2 = reader[1].ToString();
      tmp3 = reader[2].ToString();
}

OdbcDataAdapter oda = new OdbcDataAdapter("select * from customer ", MyConnection);
DataSet ds = new DataSet();

oda.Fill(ds, "employee");
DataGridView dgv = new DataGridView();
dgv.DataSource = ds.Tables["employee"];
MyConnection.Close();
```

Pros & Cons of JDBC/ODBC reading

- JDBC/ODBC reading
  - Advantages:
    - Good performance, especially for accessing massive data
    - Take advantage of various functions provided by DBMS
    - Use stored procedures to implement complex logics
  - Disadvantages:
    - Coupling with DBMS
    - Coupling with data structure
    - Programming is complicate
  - How to avoid the disadvantages?

O/R Mapping

- Working with both Object-Oriented software and Relational Databases can be
  - cumbersome and time consuming.
- Development costs are significantly higher due to
  - a paradigm mismatch between how data is represented in objects versus relational databases.
- The term Object/Relational Mapping refers to
  - the technique of mapping data from an object model representation to a relational data model representation (and visa versa)

Hibernate – first class

REin

REliable, INtelligent & Scalable Systems

- User.class

```
public class User {
    private Long id;

    private String username;
    private String password;
    private String email;

    public User() {}

    public Long getId() {  return id; }
    public void setId(Long id) {  this.id = id;  }

    public String getUsername() {  return username;  }
    public void setUsername(String username) {  this.username = username;  }

    public String getPassword() {  return password;  }
    public void setPassword(String password) {  this.password = password;  }

    public String getEmail() {  return email;  }
    public void setEmail(String email) { this.email = email; }

}
```

Hibernate – mapping

REliable, INtelligent & Scalable Systems

- User.hbm.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
        "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

<hibernate-mapping package="sample">

    <class name="User" table="tbl_user">
        <id name="id" column="id">
            <generator class="native"/>
        </id>
        <property name="username" />
        <property name="password"/>
        <property name="email"/>
    </class>

</hibernate-mapping>
```

Hibernate – configuration

REliable, INtelligent & Scalable Systems

- hibernate.cfg.xml

```
<hibernate-configuration>
    <session-factory>
        <property name="connection.driver_class">
                com.mysql.jdbc.Driver
        </property>
        <property name="connection.url">
                jdbc:mysql://localhost:3306/sample_one
        </property>
        <property name="connection.username">root</property>
        <property name="connection.password">12345678</property>
        <property name="connection.pool_size">1</property>
        <property name="dialect">
                org.hibernate.dialect.MySQL5Dialect
        </property>
        <property name="current_session_context_class">thread</property>
```

Hibernate – configuration

REliable, INtelligent & Scalable Systems

```
        <property name="cache.provider_class">
                org.hibernate.cache.internal.NoCacheProvider
        </property>

        <!-- Echo all executed SQL to stdout -->
        <property name="show_sql">true</property>

        <!-- Drop and re-create the database schema on startup -->
        <property name="hbm2ddl.auto">update</property>
        <mapping resource="sample/User.hbm.xml"/>
    </session-factory>
</hibernate-configuration>
```

Helper class

REliable, INtelligent & Scalable Systems

```
public class HibernateUtil {
    private static final SessionFactory sessionFactory = buildSessionFactory();
    private static SessionFactory buildSessionFactory() {
        try {
            // Create the SessionFactory from hibernate.cfg.xml
            return new Configuration().configure().buildSessionFactory(); }
        catch (Throwable ex) {
                      // Make sure you log the exception, as it might be
            System.err.println("Initial SessionFactory creation failed." + ex);
                      throw new ExceptionInInitializerError(ex);
        }
    }
    public static SessionFactory getSessionFactory() {
        return sessionFactory;
    }
}
```

Servlet

REin

REliable, INtelligent & Scalable Systems

```
public class UserManagerServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;
    public UserManagerServlet() {
        super();
    }

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
                     throws ServletException, IOException {
         try {
                 HibernateUtil.getSessionFactory().
                         getCurrentSession().beginTransaction();
                 PrintWriter out = response.getWriter();
                 out.println("<html><head><title>
                                 User Manager</title></head><body>");
                 listUser(out);

                 out.println("</body></html>");
                 out.flush();
                 out.close();
                 HibernateUtil.getSessionFactory().
                         getCurrentSession().getTransaction().commit();
        }
```

Servlet

REliable, INtelligent & Scalable Systems

```
        catch (Exception ex) {
            HibernateUtil.getSessionFactory()
                .getCurrentSession().getTransaction().rollback();
            if ( ServletException.class.isInstance( ex ) ) {
                throw ( ServletException ) ex;
            }
            else {
                throw new ServletException( ex );
            }
        }
}
```

Servlet


```
private void listUser(PrintWriter out) {
   List result = HibernateUtil.getSessionFactory()
           .getCurrentSession().createCriteria(User.class).list();
   if (result.size() > 0) {
       out.println("<h2>Users in database:</h2>");
       out.println("<table border='1'>");
       out.println("<tr>");
       out.println("<th>id</th>");
       out.println("<th>Username</th>");
       out.println("<th>Password</th>");
       out.println("<th>email</th>");
       out.println("</tr>");
       Iterator it = result.iterator();
       while (it.hasNext()) {
           User user = (User) it.next();
           out.println("<tr>");
           out.println("<td>" + user.getId() + "</td>");
           out.println("<td>" + user.getUsername() + "</td>");
           out.println("<td>" + user.getPassword() + "</td>");
           out.println("<td>" + user.getEmail() + "</td>");
           out.println("</tr>");
       }
       out.println("</table>");
   }
}
```

Pros and Cons of ORM

REliable, INtelligent & Scalable Systems

- ORM
  - Advantages:
    - Independent of DBMS
    - Independent of data structure
    - OOP
  - Disadvantages:
    - Impact on performance
    - Can NOT utilize the extra functions in addition to standard functions defined in specification.
    - Mapping between O and R maybe complex.
    - Can NOT invoke stored procedures.

Which one is better ?

- JDBC/ODBC reading or ORM?
  - Requirements Driven
  - Requirement 1:
    - To list all records about courses selection of a certain semester.
    - It could be a big records set with over 150,000 records selected from millions of records
  - Requirement 2:
    - To get the average age of all undergraduate students
  - Requirement3:
    - To ensure that the database could be moved from ORACLE to DB2

Project

- To develop a Book Store with PHP
  - jQuery UI
  - MySQL DB
  - Apache HTTP Server
  - To implement the CRUD of book, order and user.
  - You can design the structure of book, order and user tables as your will.

References

- Java 6 RowSet 使用完全剖析
  - http://www.ibm.com/developerworks/cn/java/j-lo-java6rowset/
- Introduction to ODBC,
  - http://msdn.microsoft.com/en-us/library/ms715408(v=vs.85).aspx
- HIBERNATE - Relational Persistence for Idiomatic Java,
  - http://docs.jboss.org/hibernate/orm/4.1/manual/en-US/html_single/#preface

REliable, INtelligent & Scalable Systems

Thank You!