

# Database Application Development using JDBC

EECS 484

Winter 09

# Overview

- Steps in an Application interacting with a database system
  - Databases accept queries, return *lists* of results
  - Applications must
    - Create queries
    - Submit them to the DB
    - Process the results (usually a list)
    - Handle errors

# JDBC: Java interface to SQL DBMS

Steps to submit a database query:

- Load the JDBC driver
- Connect to the data source (DBMS)
- Execute SQL statements

# JDBC Driver Management

- All drivers are managed by the `DriverManager` class
- Loading a JDBC driver:
  - In the Java code:  
`Class.forName("oracle.jdbc.driver.OracleDriver");`
  - When starting the Java application:  
`-Djdbc.drivers=oracle/jdbc.driver`

# Connections in JDBC

- We interact with a data source through sessions. Each connection identifies a logical session.
- JDBC URL:  
jdbc:<subprotocol>:<otherParameters>

## Example:

```
String url = "jdbc:oracle:thin:@db8.engin.umi ch. edu: 1521: muscle";  
Connection con;  
try{  
    con = DriverManager.getConnection(url, userI d, password);  
} catch (SQLException e) {...}
```

# A (Semi-)Complete Example

```
Connection con = // connect
DriverManager.getConnection(url, "login", "pass");

Statement stmt = con.createStatement(); // set up stmt
String query = "SELECT name FROM Sailors where rating = 0.1";
ResultSet rs = stmt.executeQuery(query);

try
{ // handle exceptions
  // loop through result tuples
  while (rs.next()) {           // rs is called a cursor
    String s = rs.getString("name");
    System.out.println(s + " ");
  }
  rs.close();                 // releases driver resources
  stmt.close();               // releases driver resources
  con.close();                // releases driver resources
} catch(SQLException ex) {
  System.out.println(ex.getMessage()+ ex.getSQLState() + ex.getErrorCode());
  // need to also release resources
}
```

# Multiple queries of the same format

```
String query = "SELECT name FROM Sailors where rating = 0.1";
```

```
ResultSet rs = stmt.executeQuery(query);
```

```
...
```

```
...
```

```
...
```

```
query = "SELECT name FROM Sailors where rating = 0.2";
```

```
ResultSet rs = stmt.executeQuery(query);
```

# How is the query processed?

- Application creates query
- Application sends query (delay)
- DBMS receives query
- Compiles and optimizes the query (delay)
- Computes the results
- Results must be marshalled into a list by a driver

# How is the query processed?

- Application creates query
- Application sends query (delay)
- DBMS receives query
- **Compiles and optimizes the query (delay)**
- Computes the results
- Results must be marshalled into a list by a driver

**Can we avoid recompilation/optimization every time?**

# Use PreparedStatement instead (precompiled)

```
String query = "SELECT name FROM Sailors where rating = ?";  
PreparedStatement st = conn.prepareStatement(query); // precompile
```

```
st.setFloat(1, 0.1);  
ResultSet rs = st.executeQuery();  
// process resultset rs for rating = 0.1
```

```
st.clearParameters();  
st.setFloat(1, 0.2);  
ResultSet rs = st.executeQuery();  
// process resultset rs for rating = 0.2
```

```
...
```

This is also called a **dynamic SQL query**.

# Another example

```
String sql = "INSERT INTO Sailors VALUES(?, ?)";
PreparedStatement pstmt = con.prepareStatement(sql);
pstmt.clearParameters();
pstmt.setString(1, sname);
pstmt.setFloat(2, rating);

// we know that no rows are returned, thus we use
executeUpdate()
int numRows = pstmt.executeUpdate();
```

# ResultSets

- `Statement.executeUpdate` only returns the number of affected records
- `Statement.executeQuery` returns data, encapsulated in a `ResultSet` object (a cursor)

```
ResultSet rs=pstmt.executeQuery(sql);
```

```
// rs is now a cursor
```

```
while (rs.next()) {  
    // process the data  
}
```

# ResultSet

A ResultSet is a very powerful cursor:

- `previous()`: moves one row back
- `absolute(int num)`: moves to the row with the specified number
- `relative(int num)`: moves forward or backward
- `first()` and `last()`

# Matching Java and SQL Data Types

SQL Type	Java class	ResultSet get method
BIT	Boolean	getBoolean()
CHAR	String	getString()
VARCHAR	String	getString()
DOUBLE	Double	getDouble()
FLOAT	Double	getDouble()
INTEGER	Integer	getInt()
REAL	Double	getFloat()
DATE	java.sql.Date	getDate()
TIME	java.sql.Time	getTime()
TIMESTAMP	java.sql.TimeStamp	getTimestamp()

# Commits and Rollbacks

By default a Connection object is in auto-commit mode

- `connection.setAutoCommit(true)`:
  - Each submitted query commits immediately
- `connection.setAutoCommit(false)`:
  - Queries need to be explicitly committed or rolled back.

# Commit/rollback structure:

```
try {  
    connection.setAutoCommit(false);  
    // do queries (one or more)  
    ...  
    connection.commit();  
} catch (java.sql.SQLException e) {  
    connection.rollback();  
}
```

# IMPORTANT

Change line in FileFinder.java:

```
public static String url =  
    "jdbc:oracle:thin:@db3.engin.umich.edu:1521:engin";
```