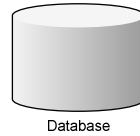


## CODE Session – JDBC

By Aaron Korver

JDBC is used to retrieve database information within Java



Developers struggle with the complexity of the JDBC API



Incorrect use of JDBC can cause application problems



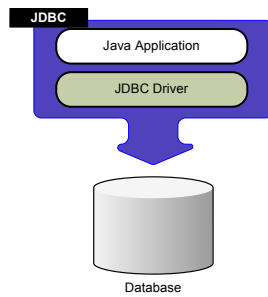
Understanding the JDBC API will help you prevent these problems



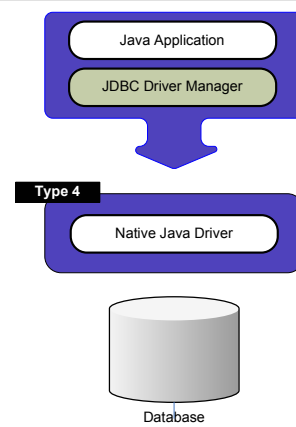
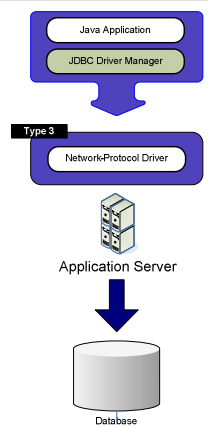
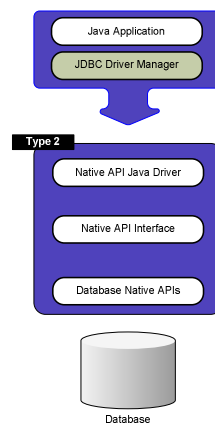
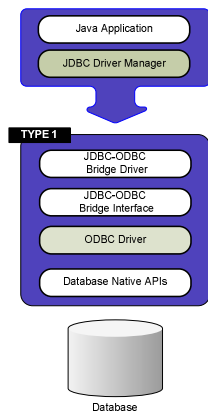
Follow along as we journey into the depths of JDBC



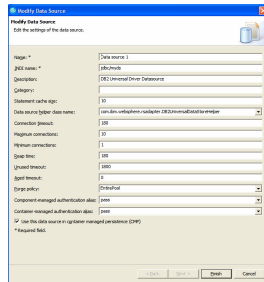
## Basic JDBC



## Drivers

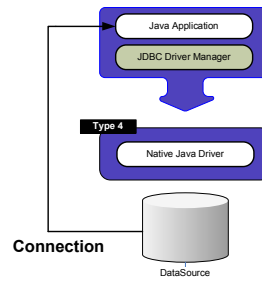


## Datasources



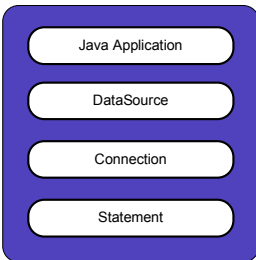
```
public DataSource getDataSource() {
    try{
        InitialContext ic = new InitialContext();
        DataSource ds =
            (DataSource)ic.lookup("jdbc/myds");
    }
    catch(NamingException ex){
        logger.error("Can't lookup
        datasource jdbc/myds");
    }
}
```

## Connection



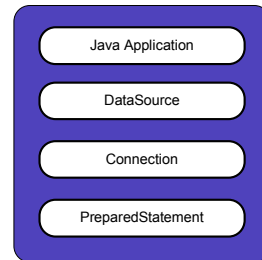
```
public String getName(){
    Connection conn = null;
    try{
        conn = getDataSource().getConnection();
        ....
    }catch(SQLException sqlExc){
        logger.error("Problem getting
        name.",sqlExc);
    }
    finally{
        try{
            if(conn != null) { conn.close(); }
        } catch(SQLException sqlExc){
            logger.error("Problem closing
            connection",sqlExc);
        }
    }
}
```

## Statement



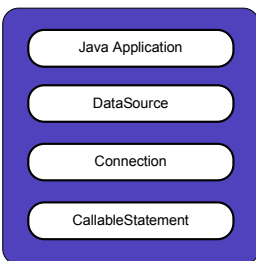
```
public void callDatabase(){
    Connection conn = null;
    Statement stmt = null;
    try{
        ....
        stmt = conn.createStatement();
        ....
    }catch(SQLException sqlExc){
        logger.error("Problem getting name.",sqlExc);
    }
    finally{
        try{
            if(stmt != null) { stmt.close(); }
        } catch(SQLException sqlExc){
            logger.error("Problem closing
            statement",sqlExc);
        }
    }
    //conn close code goes here
}
```

## PreparedStatement



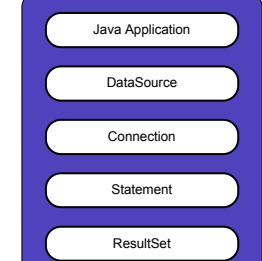
```
public void callDatabase(){
    Connection conn = null;
    PreparedStatement stmt = null;
    try{
        ....
        String sql = "Select name from table where id=?";
        stmt = conn.prepareStatement(sql);
        stmt.setString(1,"00002321");
        ....
    }catch(SQLException sqlExc){
        logger.error("Problem getting name.",sqlExc);
    }
    finally{
        try{
            if(stmt != null) { stmt.close(); }
        } catch(SQLException sqlExc){
            logger.error("Problem closing
            statement",sqlExc);
        }
    }
    //conn close code goes here
}
```

## CallableStatement



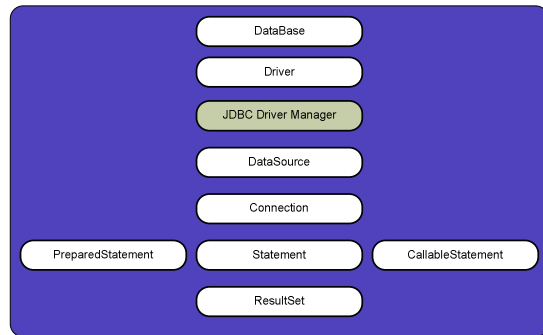
```
public void callDatabase(){
    Connection conn = null;
    CallableStatement stmt = null;
    try{
        ....
        String sql = "{? = CALL MYSP(?)}";
        stmt = conn.prepareCall(sql);
        stmt.registerOutParameter(1,Types.String);
        stmt.setString("00002321");
        ....
    }catch(SQLException sqlExc){
        logger.error("Problem getting name.",sqlExc);
    }
    finally{
        try{
            if(stmt != null) { stmt.close(); }
        } catch(SQLException sqlExc){
            logger.error("Problem closing statement",sqlExc);
        }
    }
    //conn close code goes here
}
```

## ResultSet

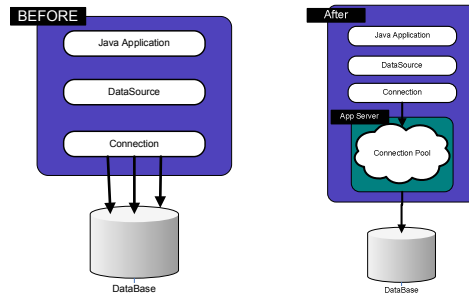


```
public void callDatabase(){
    Connection conn = null;
    PreparedStatement stmt = null;
    ResultSet rs = null;
    try{
        ....
        String sql = "Select name from table where id=?";
        ....
        logger.debug("Executing statement [" + sql + "]");
        rs = stmt.executeQuery();
        while(rs.next()) {
            name = rs.getString("name");
        }
    }catch(SQLException sqlExc){
        logger.error("Problem getting name.",sqlExc);
    }
    finally{
        try{
            if(rs != null) { rs.close(); }
        } catch(SQLException sqlExc){
            logger.error("Problem closing resultSet",sqlExc);
        }
    }
    //stmt close code goes here
    //conn close code goes here
}
```

## JDBC Summary



## Connection Pooling



## JDBC “Best” Practices

- Close resources in the same place you created them.
- Batch insert/update.
- try/catch/finally
- Use a connection pool
- Don't use SELECT \*
- Use PreparedStatements
- Use get\*() instead of getObject()

## Close your resources!

```

finally
{
    try {
        if(rs != null) {rs.close();}
    } catch (SQLException e3) {
        logger.error("Problem closing ResultSet",e3);
    }
    try {
        if(ps != null) {ps.close();}
    } catch (SQLException e1) {
        logger.error("Problem closing Statement",e3);
    }
    try {
        if(conn != null) {conn.close();}
    } catch (SQLException e2) {
        logger.error("Problem closing Connection",e3);
    }
}
  
```

## Don't Repeat Yourself (D.R.Y)

```

public class DBUtil {
    public static void close(Connection conn) {
        try {
            if(conn != null) {conn.close();}
        } catch (SQLException e2) {
            logger.error("Problem closing Connection",e3);
        }
    }
    public static void close(Statement stmt) {
        try {
            if(stmt != null) {stmt.close();}
        } catch (SQLException e1) {
            logger.error("Problem closing Statement",e3);
        }
    }
    public static void close(ResultSet rs) {
        try {
            if(rs != null) {rs.close();}
        } catch (SQLException e3) {
            logger.error("Problem closing ResultSet",e3);
        }
    }
}

Now your code becomes....
finally {
    DBUtil.close(resultSet);
    DBUtil.close(statement);
    DBUtil.close(connection);
}
  
```

## Batch Insert/Update Example

```

PreparedStatement stmt = con.prepareStatement( "INSERT INTO
employees VALUES (?, ?)");

stmt.setInt(1, 2000);
stmt.setString(2, "Kelly Kaufmann");
stmt.addBatch();

stmt.setInt(1, 3000);
stmt.setString(2, "Bill Barnes");
stmt.addBatch();

// submit the batch for execution
int[] updateCounts = stmt.executeBatch();
  
```

## Advanced Tip for Generated Keys

```
String SQL = "INSERT INTO.....";
pstmt = conn.prepareStatement(SQL,
Statement.RETURN_GENERATE_KEYS);
.....
ps.execute();
ResultSet keys = ps.getGeneratedKeys();
while(keys.next())
    logger.info(keys.getBigDecimal(1));
.....
```

# BONUS Material

## Phantom Reads

| Id | Name | Age |
|----|------|-----|
| 1  | Joe  | 20  |
| 2  | Jill | 25  |

/\* Transaction 1 \*/

/\* Transaction 2 \*/

/\* Query 1 \*/

SELECT \* FROM users WHERE  
age BETWEEN 10 AND 30;

/\* Query 2 \*/

INSERT INTO users VALUES  
( 3, 'Bob', 27 );  
COMMIT;

/\* Query 1 \*/

SELECT \* FROM users WHERE  
age BETWEEN 10 AND 30;

## Non-Repeatable Read

| Id | Name | Age |
|----|------|-----|
| 1  | Joe  | 20  |
| 2  | Jill | 25  |

/\* Transaction 1 \*/

/\* Transaction 2 \*/

/\* Query 1 \*/

SELECT \* FROM users WHERE  
id = 1;

/\* Query 2 \*/

UPDATE users SET age = 21  
WHERE id = 1;  
COMMIT;

/\* Query 1 \*/

SELECT \* FROM users WHERE  
id = 1;

## Dirty Read

| Id | Name | Age |
|----|------|-----|
| 1  | Joe  | 20  |
| 2  | Jill | 25  |

/\* Transaction 1 \*/

/\* Transaction 2 \*/

/\* Query 1 \*/

SELECT \* FROM users WHERE  
id = 1;

/\* Query 2 \*/

UPDATE users SET age = 21  
WHERE id = 1;

/\* Query 1 \*/

SELECT \* FROM users WHERE  
id = 1;

## ANSI Isolation Levels

| Isolation Level  | Dirty Read | Non-repeatable Read | Phantom Read |
|------------------|------------|---------------------|--------------|
| READ UNCOMMITTED | Permitted  | Permitted           | Permitted    |
| READ COMMITTED   | --         | Permitted           | Permitted    |
| REPEATABLE READ  | --         | --                  | Permitted    |
| SERIALIZABLE     | --         | --                  | --           |

## Setting the Isolation Level

The screenshot shows the 'References' configuration window in Visual Studio. The window is titled 'References' and contains a list of resources on the left and configuration options on the right. The resource 'Resource1' is selected. The configuration options include 'Name', 'Description', 'Type', 'Authentication', 'Authentication Mode', 'Use Custom Login Configuration', 'Login Configuration Name', 'Properties', 'Property', 'Value', 'Description', 'Workgroup Extensions', 'Isolation level', and 'Connection policy'. The 'Isolation level' dropdown is set to 'READ-ONLY' and is circled in red. The 'Connection policy' dropdown is set to 'Default'. The 'Workgroup Extensions' section is also visible, showing a list of extensions.

References

This web application references the following resources:

- Resource1

Name: Resource1

Description:

Type: Service and Data Source

Authentication: Container

Authentication Mode:

Use Custom Login Configuration:

Login Configuration Name:

Properties:

Property:

Value:

Description:

Workgroup Extensions:

The following extensions are used for this Web Service:

Isolation level: READ-ONLY

Connection policy: Default

Buttons: Add, Remove

Navigation: Overview | Services | Filter | Security | References | WSS Handler | Pages | Variables | WSS Extension | WSS Extension | WSS Extension | Source