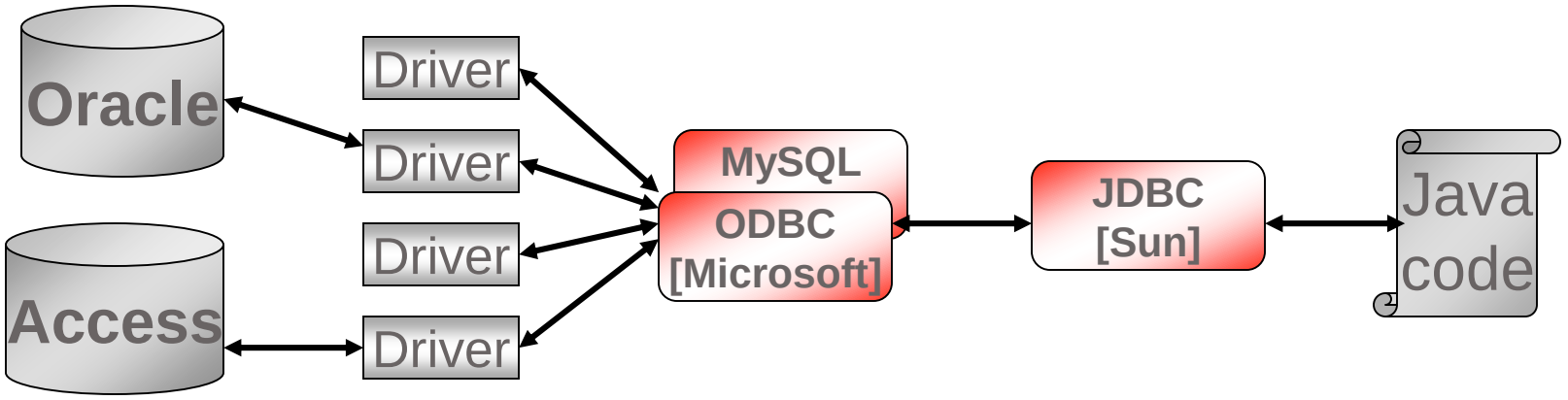
hi-tech college + JOHN BRYCE Leading in IT Education a matrix company

JDBC

Java Data-Base Connection

JDBC

- An interface that contain a flexible D.B connection
- ODBC - JDBC bridge (or others like MySQL):



Using JDBC

1. Loading JDBC Driver
2. Establishing Connection
3. Creating a Statement (for execute & update queries)
4. Process the results (ResultSet)
5. Closing connection

Main Classes

- **DriverManager** - Contain the Drivers list
- **DataSource -** A driver representation [javax.sql.DataSource]
- **Connection** - The connection properties class
- **Statement** - The queries carrier class [DML, Select, Create, Drop]
- **ResultSet** - Results class. acts like a cursor

STEP 1

Driver Loading

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

Class.forName("")

this static method load an instance of the requested Class. In this case it's sun provided class that will be used later by the **DriverManager** by a **static block**

DriverManager

the DriverManager holds a list of all odbc-jdbc drivers that were located

Request a specified driver (located in the server directory) using JNDI :

```
Hashtable env = new Hashtable(11);
env.put(Context.INITIAL_CONTEXT_FACTORY,"weblogic.jndi.WLInitialContextFactory");
env.put(Context.PROVIDER_URL, "http://localhost:7001");
Context ctx=new InitialContext(env);
javax.sql.DataSource ds=(javax.sql.DataSource)ctx.lookup("weblogic.jdbc.JdbcServices");
```

Driver Loading

- There are more ways:

```
sun.jdbc.odbc.JdbcOdbcDriver driver = new sun.jdbc.odbc.JdbcOdbcDriver();
```

Less portable

```
System.setProperty("jdbc.drivers","sun.jdbc.odbc.JdbcOdbcDriver");
```

Better…

STEP 2 Connecting

```
Connection con=DriverManager.getConnection("jdbc:odbc:myDB");
```

or

```
Connection con=DriverManager.getConnection("jdbc:odbc:myDB","user","password");
```

Connection

The connection manager

• connection status	boolean isClosed ()
• close connection	close ()
• statement producer	Statement createStatement ()
• commit	setAutoCommit(boolean) , commit()

Connecting a specific driver represented by the DataSource:

```
javax.sql.DataSource ds=(javax.sql.DataSource)ctx.lookup("weblogic.jdbc.JdbcServices");
Connection con= ds.getConnection();
```

STEP 3

Submitting Queries

- Creating Statement:

```
Statement stmt =con.createStatement();
```

- D.B Updates:

```
stmt.executeUpdate(" create table book (name VARCHAR2(15),  cost NUMBER(4)) ");
stmt.executeUpdate("insert into book values (' "+name+" ',"+cost+") ");
stmt.executeUpdate(" update book set cost=400 where name='JAVA2' ");
stmt.executeUpdate("delete from book where cost>100");
stmt.executeUpdate(" drop table book  ");
```

STEP 4

Submitting Queries

- Query Statement:

```
try{
    ResultSet rs = stmt.executeQuery("select * from book");
    while(rs.next()){
        String tempName=rs.getString("name");
        int tempCost=rs.getInt("cost");
        System.out.println(tempName+" "+tempCost);
    }
}catch(SQLException general){
        System.out.println("Select Failed");
}
```

ResultSet

- holds the result records of the query
- parses retrieved data-types:

boolean getBoolean (String colName)
int getInt (String colName)
String getString (String colName)
Date getDate (String colName)
float getFloat (String colName)

STEP 5

Closing Connection

```
con.close();
```

can be under the finally case:

```
…any query operation
...
}catch(SQLException e){
            e.printStackTrace();
}finally{
      try{
                con.close();
      }catch(Exception ex){…}
}
```

For any appropriate Driver the **con.close();** method should be called

Prepared Statements

- Every statement sent to the Data-Base is precompiled to a Prepared-Statement
- Getting the prepared statement will enable to execute the same statement many times
- Extends Statement interface

Prepared Statements

- Using prepared statements :

```
PreparedStatement pstmt =
con.prepareStatement("UPDATE book SET price = ? WHERE ID = ?");

 pstmt.setFloat(1, 153.00);
 pstmt.setInt(2, 110592);

pstmt.execute();
```

CallableStatement

- Enables SQL Stored Procedures execution
- May register an OUT parameter (when return values expected)
- Extends PreparedStatements Interface

ResultSetMetaData

- Enable to query the ResultSet structure

- Taken from a ResultSet instance (after executing query)

```
ResultSet rs = stmt.executeQuery("SELECT * FROM book_table");
ResultSetMetaData rsmd = rs.getMetaData();
int numberOfColumns = rsmd.getColumnCount();
Sring name = rsmd.getColumnName(1);
int type = rsmd.getColumnType(1);
```

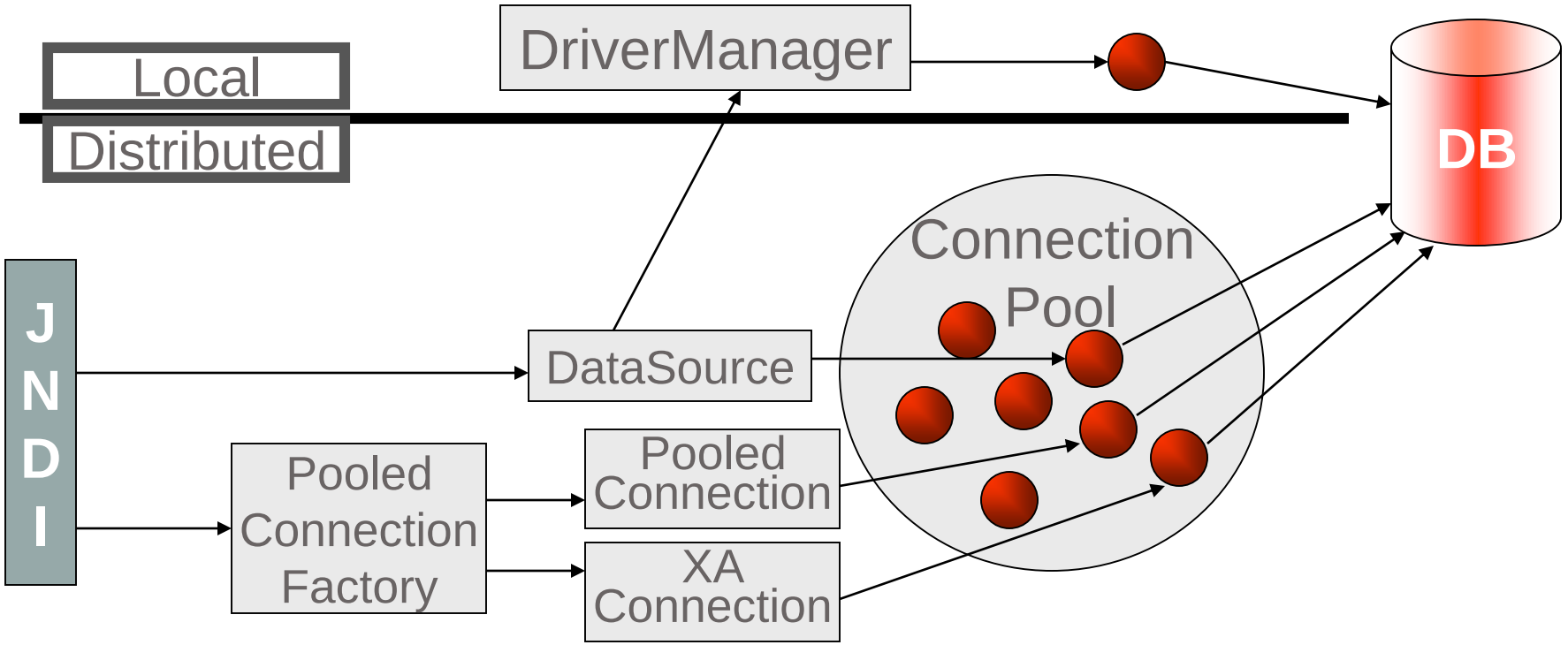
java.sql.Types
- represents DB types as integers

JDBC 2 Improvements

- Connection Pooling
- 2 Phase Commit XA Standard support
- RowSet - Use of JavaBeans as Value-Objects
- JNDI mapping enabled

Connection Pooling

- Used for the use of multiple connection in enterprise applications (like Application Servers)

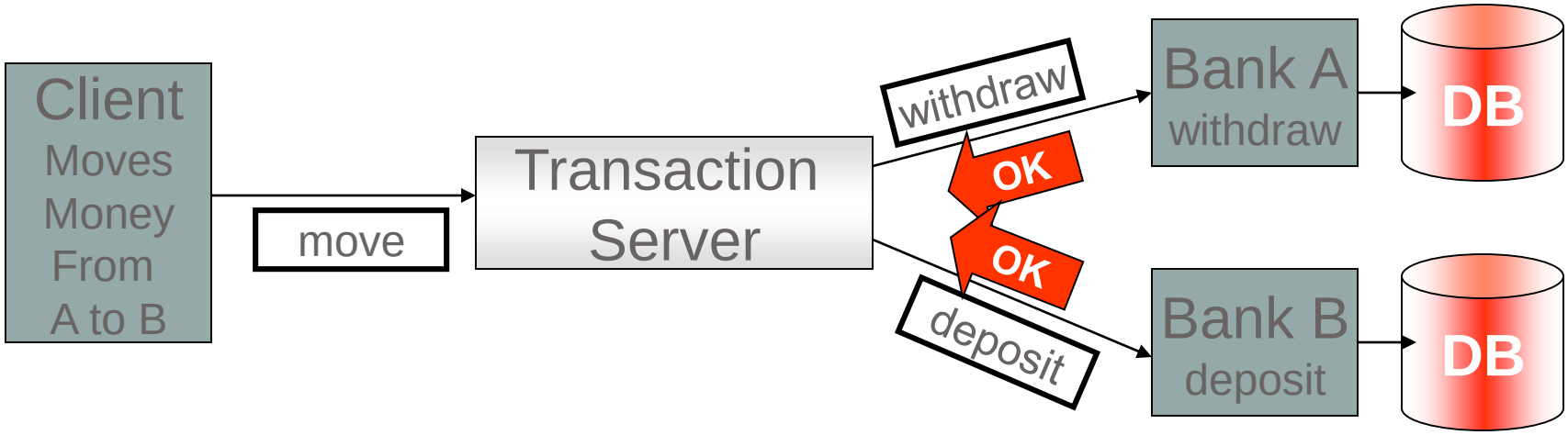


2 Phase Commit

- In distributed Systems different connections are used
  - even on different DataBases
- A Transaction Server manages the whole process Tx
- XA Standards for 2 Phase Commit is supported by:
  - XAResourse
  - XAConnection interfaces

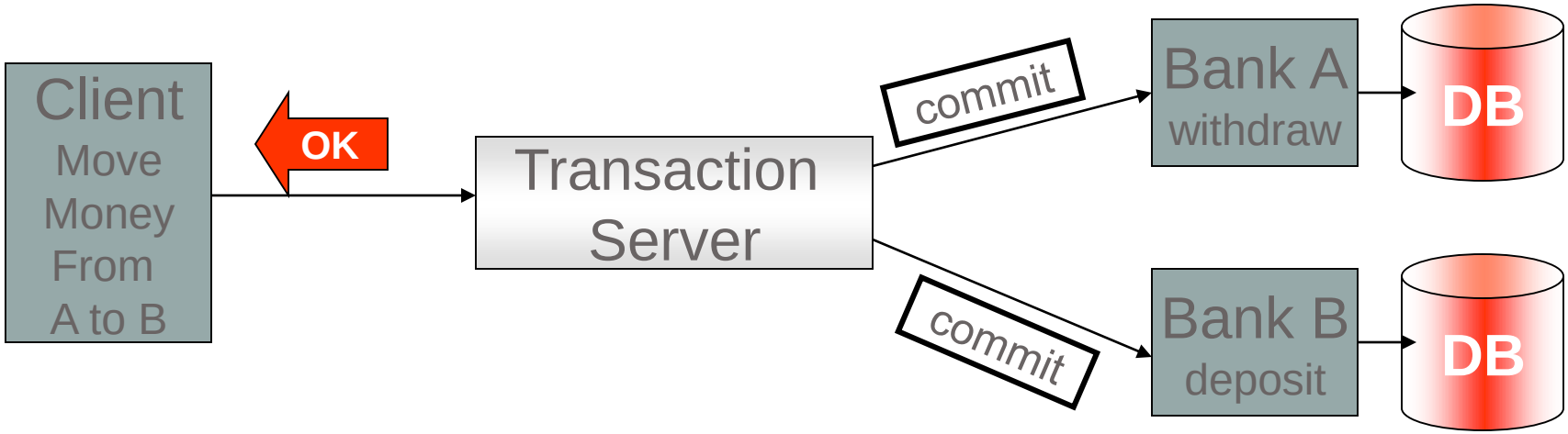
2 Phase Commit

Phase 1



2 Phase Commit

Phase 2



References

- http://java.suncom/products/jdbc
- SUN Educational Services SL-301