Lecture # 6

Imran Ahsan

Database Connectivity

JDBC

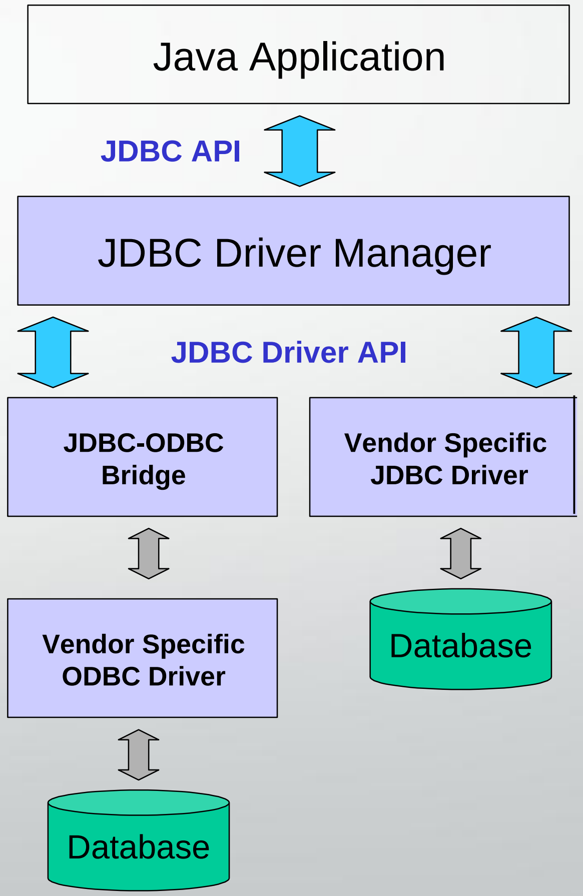
Introduction

- JDBC provides a standard library for accessing relational databases
  - Way to establish connection to database
  - Approach to initiating queries
  - Method to create stored (parameterized) queries
  - The data structure of query result (table)
    - Determining the number of columns
    - Looking up metadata, etc.
- API does *not* standardize SQL syntax
- `java.sql` package contains JDBC classes

Introduction

JDBC consists of two parts:

1. JDBC API, a purely Java-based API
2. JDBC Driver Manager, which communicates with vendor-specific drivers that perform the real communication with the database



Seven Basic Steps in Using JDBC

1. Load the driver
2. Define the Connection URL
3. Establish the Connection
4. Create a Statement object
5. Execute a query
6. Process the results
7. Close the connection

Seven Basic Steps in Using JDBC

1. Load the driver
    1. Wizard Based Driver Inclusion
    2. Including driver in code

```
try {
  Class.forName("oracle.jdbc.driver.OracleDriver");
  Class.forName("com.mysql.jdbc.Driver");
} catch { ClassNotFoundException cnfe) {
  System.out.println("Error loading driver: " cnfe);
}
```

2. Define the Connection URL

```
String host = “localhost";
String dbName = "someName";
int port = 8080;
String oracleURL = "jdbc:oracle:thin:@" + host +
                   ":" + port + ":" + dbName;
String mysqlURL = "jdbc:mysql://" + host  +
                  ":" + port + "/" + dbName;
```

Seven Basic Steps in Using JDBC

3. Establish the Connection

```
String username = “root";

String password = “";

Connection connection =

    DriverManager.getConnection(mysqlURL,
                                username,
                                password);
```

Seven Basic Steps in Using JDBC

4. Create a Statement

```
Statement statement = connection.createStatement();
```

5. Execute a Query

```
String query = "SELECT col1, col2, col3 FROM sometable";
ResultSet resultSet = statement.executeQuery(query);
```

- To modify the database, use **`executeUpdate`**, supplying a string that uses **`UPDATE`**, **`INSERT`**, or **`DELETE`**

Seven Basic Steps in Using JDBC

6. Process the Result

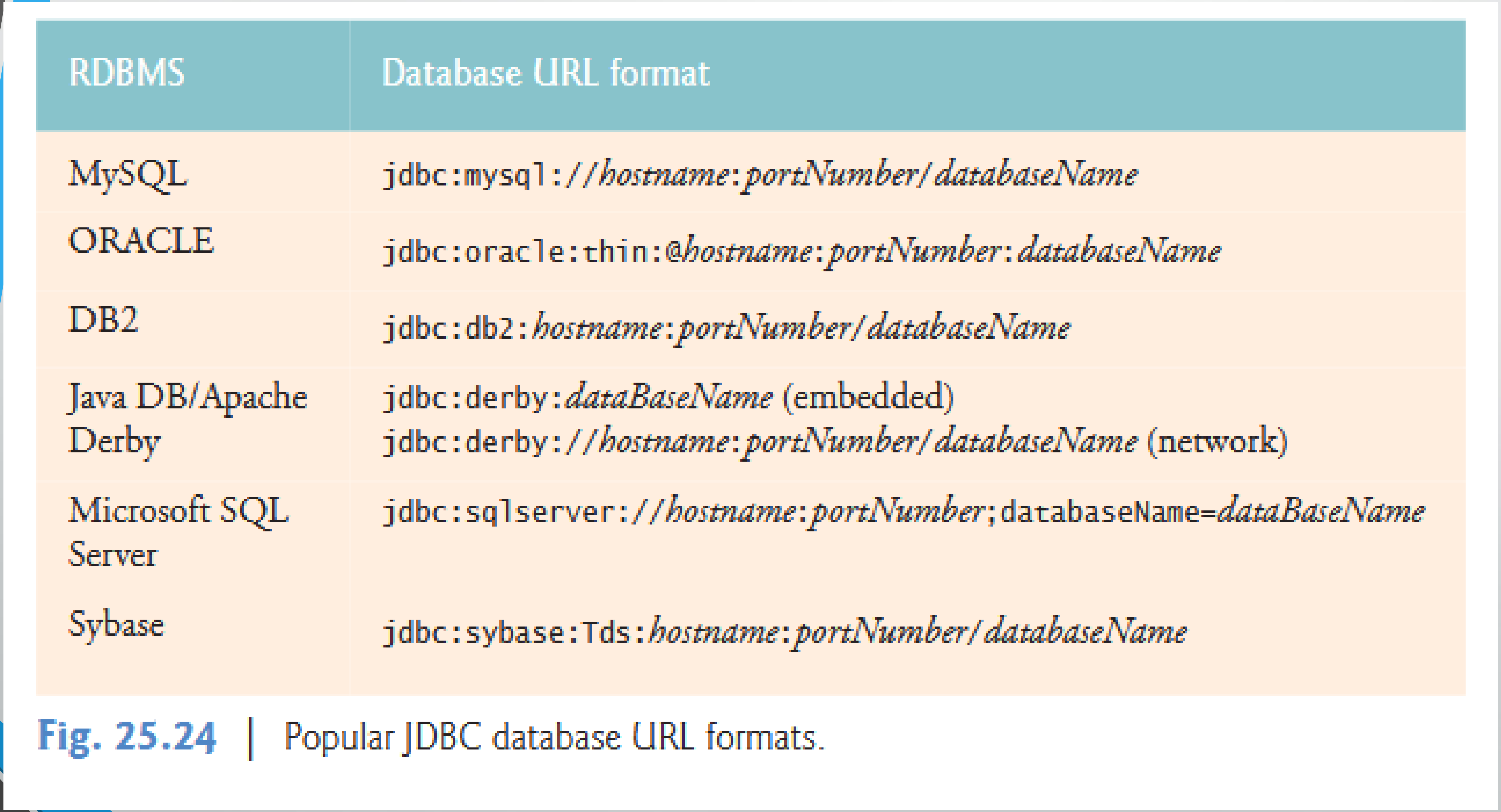
```
while(resultSet.next()) {
System.out.println(resultSet.getString(1) + " " +
                   resultSet.getString(2) + " " +
                   resultSet.getString(3));
}
```

- First column has index 1, not 0
- **`ResultSet`** provides various **`get`** methods that take a column index or name and returns the data

7. Close the Connection

```
connection.close();
```

- As opening a connection is expensive, postpone this step if additional database operations are expected



RDBMS	Database URL format
MySQL	jdbc:mysql://*hostname*:*portNumber*/*databaseName*
ORACLE	jdbc:oracle:thin:@*hostname*:*portNumber*:*databaseName*
DB2	jdbc:db2:*hostname*:*portNumber*/*databaseName*
Java DB/Apache Derby	jdbc:derby:*dataBaseName* (embedded) jdbc:derby://*hostname*:*portNumber*/*databaseName* (network)
Microsoft SQL Server	jdbc:sqlserver://*hostname*:*portNumber*;databaseName=*dataBaseName*
Sybase	jdbc:sybase:Tds:*hostname*:*portNumber*/*databaseName*

Fig. 25.24 | Popular JDBC database URL formats.

Basic JDBC Example

```
import java.sql.*;


public class TestDriver       {
   public static void main(String[] Args)        {
      try {
  Class.forName("com.mysql.jdbc.Driver").newInstance();}
      catch (Exception E) {
          System.err.println("Unable to load driver.");
          E.printStackTrace();
                }
      try {
        Connection C = DriverManager.getConnection(
        "jdbc:mysql://localhost:3307/testdb",
        "root", "xyz"); //?user=root&password=xyz");
```

Basic JDBC Example

```
Statement s = C.createStatement();
String sql="select * from table";
s.execute(sql);
    ResultSet res=s.getResultSet();
    if (res!=null)   {
            while(res.next()){//note Sql start with 1
        System.out.println("\n"+res.getString(1)
             + "\t"+res.getString(2));
            }
        }
c.close();
    }
    catch (SQLException E) {
       System.out.println("SQLException: " + E.getMessage());
       System.out.println("SQLState:     " + E.getSQLState());
       System.out.println("VendorError:  " + E.getErrorCode());
  }}}
```

`Statement` Object

- After you have a connection, you need to create a statement.
- There are three alternatives, each with plusses and minuses.

- `Statement`—used for a query that will be executed once.
- `PreparedStatement`—used for a query that will be executed multiple times
- `CallableStatement`—used for a query that executes a stored procedure.

`Statement` Object

- The **`Statement`** object is the easiest to work with.
- The **`Statement`** object is the *least* efficient.

- `String query = “SELECT * FROM MYTABLE WHERE ID = 2”;`
- `Connection con = DriverManager.getConnection( url, user, pass );`
- `Statement stmt = con.createStatement();`
- `ResultSet rs = stmt.executeQuery( query );`

PreparedStatement Object

- The **PreparedStatement** object requires more work.
- The **PreparedStatement** object is the *most* efficient.
- The query contains a question mark that is replaced.

```
String query = “SELECT * FROM MYTABLE WHERE ID = ?”;
```

- `Connection con = DriverManager.getConnection( url, user, pass );`

-
  ```
  PreparedStatement pstmt = con.prepareStatement( query );
      pstmt.setString( 1, 494 );
  ```

This line substitutes **494** for the first question mark in the query.

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

`CallableStatement` Object

- The `CallableStatement` object is only appropriate for calling a stored procedure.
- The syntax of how you call the stored procedure is database specific.

- `String call = "{ call myProcdure }";`

- `Connection con = DriverManager.getConnection( url, user, pass );`

- `CallableStatement cstmt = con.prepareCall( call );`

- `ResultSet rs = cstmt.executeQuery();`

ResultSet Object

- The ResultSet object receives the results of the query.

- String query = "SELECT COL1, COL2 FROM MYTABLE WHERE ID = 2";
- Connection con = DriverManager.getConnection(url, user, pass);
- Statement stmt = con.createStatement();
- ResultSet rs = stmt.executeQuery(query);
- while(rs.next())
- {
-     String myCol1 = rs.getString("COL1");
-     String myCol2 = rs.getString("COL2");

}

next() returns true while there are results

These correspond to columns in the original query.

`ResultSet` Object

- No matter which kind of statement you choose, the ResultSet object is used the same way.
- As with the `Connection` object, you must close your `ResultSet`!

Important

```
try
{
    String output = null;
    String query = "SELECT username from MYTABLE where pass='foo' ";
    Connection con = DriverManager.getConnection( url, us, pass);
    Statement stmt = con.createStatement();
    ResultSet rs = stmt.executeQuery( query );

    while( rs.next() )
    {
        output = rs.getString( "username" );
    }

    rs.close();
    stmt.close();
    con.close();
}
catch( SQLException sql )
{
    System.out.println( "....................." );
}
```

You must close these three items, in the reverse order that you opened them!

End.