JDBC Driver Types

- The JDBC driver specification classifies JDBC drivers into four groups.
- Each group is referred to as a JDBC driver type and addresses a specific need for communicating with various DBMSs

Type 1 JDBC to ODBC Driver

- Type 1 creates the both JDBC and ODBC(Open data base connectivity) drivers.
- Both ODBC and JDBC have similar driver specifications and an API.
- The JDBC to ODBC driver is used to translate DBMS calls between the JDBC specification and the ODBC specification.
- The JDBC to ODBC driver receives messages from a J2ME application that conforms to the JDBC specification
- Those messages are translated by the JDBC to ODBC driver into the ODBC message format, which is then translated into the message format understood by the DBMS.
- The extra translation might negatively impact performance.
- This type of driver is recommended only for experimental use or when no other alternative is available.

Type 2 Java/Native Code Driver

- The Java/Native Code driver uses Java classes to generate platform-specific code — that is, code only understood by a specific DBMS.
- The manufacturer of the DBMS provides both the Java/Native Code driver and API classes so the J2ME application can generate the platform-specific code
- Type 2 driver, JDBC API calls are converted into native C/C++ API calls, which are unique to the database.
- These drivers are typically provided by the database vendors and used in the same manner as the JDBC-ODBC Bridge.
- The vendor-specific driver must be installed on each client machine.
- Disadvantage of using a Java/ Native Code driver is the loss of some portability of code.
- The Oracle Call Interface (OCI) driver is an example of a Type 2 driver.

Type 3 JDBC Driver

- The Type 3 JDBC driver, also referred to as the Java Protocol.
- The Type 3 JDBC driver converts SQL queries into JDBC-formatted statements.
- The JDBC-formatted statements are translated into the format required by the DBMS.
- In a Type 3 driver, a three-tier approach is used to access databases.
- The JDBC clients use standard network sockets to communicate with a middleware application server.
- The socket information is then translated by the middleware application server into the call format required by the DBMS, and forwarded to the database server.
- This kind of driver is extremely flexible, since it requires no code installed on the client and a single driver can actually provide access to multiple databases.

- Type 4 JDBC Driver
- The Type 4 JDBC driver is also known as the Type 4 Database Protocol.
- SQL queries do not need to be converted to JDBC-formatted systems. This is the fastest way to communicate SQL queries to the DBMS.
- In a Type 4 driver, a pure Java-based driver communicates directly with the vendor's database through socket connection.
- This is the highest performance driver available for the database and is usually provided by the vendor itself.
- This kind of driver is extremely flexible, you don't need to install special software on the client or server. Further, these drivers can be downloaded dynamically.
- Ex:MySQL's Connector/J driver is a Type 4 driver.

Which Driver should be Used?

- If you are accessing one type of database, such as Oracle, Sybase, or IBM, the preferred driver type is 4.
- If your Java application is accessing multiple types of databases at the same time, type 3 is the preferred driver.
- Type 2 drivers are useful in situations, where a type 3 or type 4 driver is not available yet for your database.
- The type 1 driver is not considered a deployment-level driver, and is typically used for development and testing purposes only.

- JDBC packages
  - JDBC API is contained in two packages.

1. java.sql contains core JDBC interfaces that provide the basics for connecting to the DBMS and interacting with data stored in the DBMS. java.sql is part of the J2SE.
2. javax.sql, is the JDBC interface that interacts with Java Naming and Directory Interface (JNDI) and manages connection pooling, among other advanced JDBC features.

- Overview of the JDBC Process
- The process of interacting the J2ME application with DBMS is divided into 5 routines.

1. loading the JDBC driver,
2. connecting to the DBMS,
3. creating and executing a statement
4. processed data returned by the DBMS
5. terminating the connection with the DBMS.

Steps Involved in Basic JDBC Operations

1. Load the JDBC driver class:

```
Class.forName("driverName");
```

2. Open a database connection:

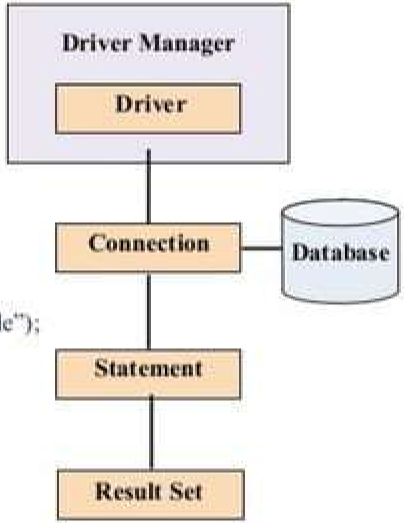
```
DriverManager.getConnection
("jdbc:xxx:datasource");
```

3. Issue SQL statements:

```
stmt = con.createStatement();
stmt.executeQuery ("Select * from myTable");
```

4. Process resultset:

```
while (rs.next()) {
name = rs.getString("name");
amount = rs.getInt("amt"); }
```



1. Load the JDBC Driver

- Load JDBC drivers before J2ME application connected to the DBMS.
- To work offline the developer must write a routine that loads the JDBC/ODBC Bridge driver called sun.jdbc.odbc.JdbcOdbcDriver.
- The driver is loaded by calling the Class.forName() method and passing it the name of the driver, as shown in the following code segment:
- Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

2. Connecting to the DBMS

- The J2ME application must connect to the DBMS using the DriverManager.getConnection() method.
- The java.sql.DriverManager class is the highest class in the java.sql hierarchy and is responsible for managing driver information.
- DriverManager.getConnection() method is passed the URL of the database, along with the user ID and password if required by the DBMS.
- The URL is a String object that contains
- The driver name and the name of the database that is being accessed by the J2ME application.
- The DriverManager.getConnection() returns a Connection interface that is used throughout the process to reference the database.
- The java.sql.Connection interface is another member of the java.sql package that manages communications between the driver and the J2ME application.
- Sends statements to the DBMS for processing
- use of the DriverManager.getConnection() method to load the JDBC/ODBC Bridge and connect to the CustomerInformation database.

```
String url = "jdbc:odbc:CustomerInformation";
String userID = "jim";
String password = "keogh";
Statement DataRequest;
private Connection Db;
try {
Class.forName( "sun.jdbc.odbc.JdbcOdbcDriver");
Db = DriverManager.getConnection(url,userID,password);
}
```

3. Creating and Executing a statement

- Send an SQL query to the DBMS for processing.
- An SQL query consists of a series of SQL commands that direct the DBMS to do something, such as return rows of data to the J2ME application.
- The Connect.createStatement() is used to create a Statement object.
- The Statement object is then used to execute a query and return a ResultSet object that contains the response from the DBMS, which is usually one or more rows of information requested by the J2ME application.

```
Statement DataRequest;
ResultSet Results;
try {
String query = "SELECT * FROM Customers";
DataRequest = Db.createStatement();
Results = DataRequest.executeQuery (query);
DataRequest.close();
}
```

4. Process Data Returned by the DBMS

- The java.sql.ResultSet object is assigned the results received from the DBMS after the query is processed.
- The java.sql.ResultSet object consists of methods used to interact with data that is returned by the DBMS to the J2ME application.
- The first time that the next() method of the ResultSet is called, the ResultSet pointer is positioned at the first row in the ResultSet and returns a boolean value.
- If false no rows in the resultSet else (true) at least one row in the resultSet.

```
ResultSet Results;
          String FirstName;
          String LastName;
          String printrow;
          boolean Records = Results.next();
          if (!Records ) {
          System.out.println( "No data returned");
          return;
          }
          else
          {
          do {
                    FirstName = Results.getString (FirstName) ;
                    LastName = Results.getString (LastName) ;
                    printrow = FirstName + " " + LastName;
                    System.out.println(printrow);
                    } while ( Results.next() );
          }
```

Terminate the Connection to the DBMS

- The connection to the DBMS is terminated by using the close() method of the Connection object once the J2ME application is finished accessing the DBMS.
- Db.close();

Database Connection

- A J2ME application does not directly connect to a DBMS.
- Instead, the J2ME application connects with the JDBC driver that is associated with the DBMS.
- the JDBC driver must be loaded and registered with the DriverManager.
- The purpose of loading and registering the JDBC driver is to bring the JDBC driver into the Java Virtual Machine (JVM) and can be used by J2ME applications.

```
try {
Class.forName( "sun.jdbc.odbc.JdbcOdbcDriver");
}
catch (ClassNotFoundException error) {
System.err.println("Unable to load the JDBC/ODBC bridge." +
error.getMessage());
System.exit(1);
}
```

The Connection

- After the JDBC driver is successfully loaded and registered, the J2ME application must connect to the database.
- The data source that the JDBC component will connect to is defined using the URL format. The URL consists of three parts:
- jdbc, which indicates that the JDBC protocol is to be used to read the URL.
- <subprotocol>, which is the JDBC driver name
- <subname>, which is the name of the database
- The getConnection() method requests access to the database from the DBMS.
- AConnection object is returned by the getConnection() method if access is granted, otherwise the getConnection() method throws an SQLException.

```
String url = "jdbc:odbc:CustomerInformation";
String userID = "jim";
String password = "keogh";
Statement DataRequest;
Connection Db;
try {
        Class.forName( "sun.jdbc.odbc.JdbcOdbcDriver");
        Db = DriverManager.getConnection(url,userID,password);     }
```

- DBMS requires information besides a user ID and password before the DBMS grants access to the database.
- This additional information is referred to as "properties" and must be associated with a Properties object, which is passed to the DBMS as a getConnection() parameter.
- Typically, properties used to access a database are stored in a text file, the contents of which are defined by the DBMS manufacturer.
- The J2ME application uses a FileInputStream object to open the file and then uses the Properties object load() method to copy the properties into a Properties object
- Ex:
- FileInputStream propFileStream = new fileInputStream("DBProps.txt");

```
 props.load(propFileStream);
Class.forName( "sun.jdbc.odbc.JdbcOdbcDriver");
 Db = DriverManager.getConnection(url, props);
```

Statement Objects

- 3 types of Statement objects is used to execute the query.
- These objects are
- Statement, which executes a query immediately;
- PreparedStatement, which is used to execute a compiled query
- CallableStatement, which is used to execute store procedures.

Statement

- The Statement object contains
- The executeQuery() which is passed the query as an argument. The query is then transmitted to the DBMS for processing.
- It returns one ResultSet object that contains rows, columns, and metadata that represent data requested by the query.
- The execute() method of the Statement object is used when multiple results may be returned.
- The executeUpdate() is used to execute queries that contain INSERT, UPDATE, DELETE, and DDL statements.,which change values in a row and remove a row, respectively. It returns an integer indicating the number of rows that were updated by the query.
- The createStatement() method of the Connection object is called to return a Statement object

Ex: executeQuery

```
String query = "SELECT * FROM
Customers";
  DataRequest = Db.createStatement();
 Results = DataRequest.executeQuery
(query);
 String query = "UPDATE Customers
```

executeUpdate

```
SET PAID='Y' WHERE BALANCE
= '0';
DataRequest = Db.createStatement();
rowsUpdated =
DataRequest.executeUpdate (query);
```

PreparedStatement

- An SQL query can be precompiled and executed by using the PreparedStatement object.
- A question mark is used as a placeholder for a value that is inserted into the query after the query is compiled. It is this value that changes each time the query is executed.
- The preparedStatement() method of the Connection object is called to return the PreparedStatement object.
- The preparedStatement() method is passed the query that is then precompiled.
- The setXXX() method of the PreparedStatement object is used to replace the question mark with the value passed to the setXXX() method.
- There are a number of setXXX() methods available in the PreparedStatement object, each of which specifies the data type of the value that is being passed to the setXXX()
- The setXXX() requires two parameters. The first parameter is an integer that identifies the position of the question mark placeholder, and the second parameter is the value that replaces the question mark placeholder.
- Ex:String query = "SELECT * FROM Customers WHERE CustNumber = ?";

```
PreparedStatement pstatement = Db.preparedStatement(query);
pstatement.setString(1, "123");
Results = pstatement.executeQuery ();
```

ResultSet

- A query is used to update, delete, and retrieve information stored in a database.
- The executeQuery() method is used to send the query to the DBMS for processing and returns a ResultSet object that contains data requested by the query.
- The ResultSet object contains methods that are used to copy data to a Java collection of objects or variable(s) for further processing.
- Data in a ResultSet object is logically organized into a virtual table consisting of rows and columns. In addition to data, the ResultSet object also contains metadata, such as column names, column size, and column data type.
- The ResultSet uses a virtual cursor to point to a row of the virtual table.
- A J2ME application must move the virtual cursor to each row, then use other methods of the ResultSet object to interact with the data stored in columns of that row.
- The virtual cursor is positioned above the first row of data when the ResultSet is returned by the executeQuery() method. This means that the virtual cursor must be moved to the first row using the next() method. The next() method returns a boolean true if the row contains data, otherwise a boolean false is returned