

Advanced JAVA

Database Connectivity

Introduction

- Business applications usually store data in the databases.
 - In most of the enterprise applications, Relational Database Management Systems (RDBMS) are used as data storage. They store the data records in tables. Each record (such as that of an employee) is represented by a table row, which consists of one or more columns or record fields (e.g., name, address, hire date). RDBMS understand the SQL language.
 - The most popular RDBMS are Oracle, DB2, Sybase, Microsoft SQL Server, and MySQL Server.
 - JDBC API is not DBMS-specific — if you write a program that uses JDBC classes to retrieve/update data in Oracle, you'll be using the same classes to work with MySQL Server or DB2. You just need the JDBC drivers from the corresponding DBMS vendor — the drivers hide their database specification behind the same public JDBC API.
-

API (Application Programming Interface)

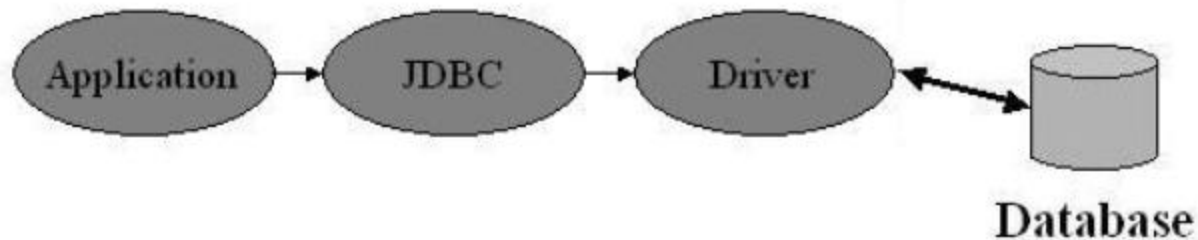
- API (Application programming interface) is a document that contains description of all the features of a product or software. It represents classes and interfaces that software programs can follow to communicate with each other. An API can be created for applications, libraries, operating systems, etc.
 - Java JDBC is a java API to connect and execute query with the database. JDBC API uses jdbc drivers to connect with the database. This API consists of a set of classes and interfaces to enable programmers to write pure Java Database applications.
 - Before JDBC, ODBC API was the database API to connect and execute query with the database. But, ODBC API uses ODBC driver which is written in C language (i.e. platform dependent and unsecured). That is why Java has defined its own API (JDBC API) that uses JDBC drivers (written in Java language)
-

JDBC VS ODBC

- The most widely used interface to access relational database today is Microsoft's ODBC API. ODBC performs similar tasks as that of JDC (Java Development Connection) and yet JDBC is preferred due to the following reasons :
 - ODBC cannot be directly used with Java because it uses a C interface. Calls from Java to native C code have a number of drawbacks in the security, implementation, robustness and automatic portability of applications.
 - ODBC makes use of pointers which have been totally removed from Java.
 - JDBC API is a natural Java Interface and is built on ODBC. JDBC retains some of the basic features of ODBC like X/Open SQL Call Level Interface.
 - JDBC is to Java programs and ODBC is to programs written in languages other than Java.
 - ODBC mixes simple and advanced features together and has complex options for simple queries. But JDBC is designed to keep things simple while allowing advanced capabilities when required.
-

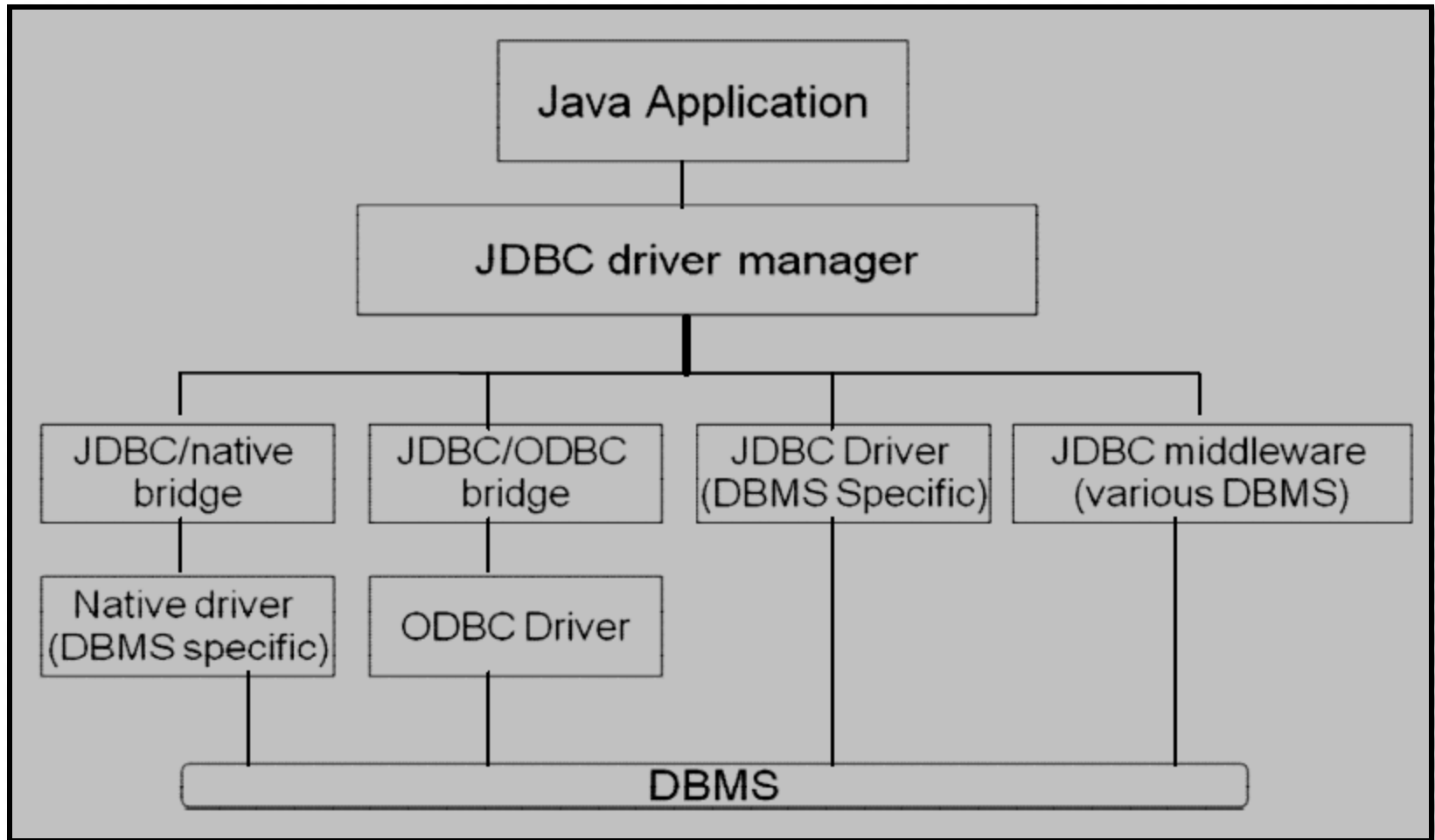
JDBC ARCHITECTURE

- Java application calls the JDBC library. JDBC loads a driver which talks to the database.



- JDBC contains three components: Application, Driver Manager, Driver.
- The user application invokes JDBC methods to send SQL statements to the database and retrieves results. JDBC driver manager is used to connect Java applications to the correct JDBC driver . JDBC driver test suite is used to ensure that the installed JDBC driver is JDBC Compliant.

JDBC ARCHITECTURE

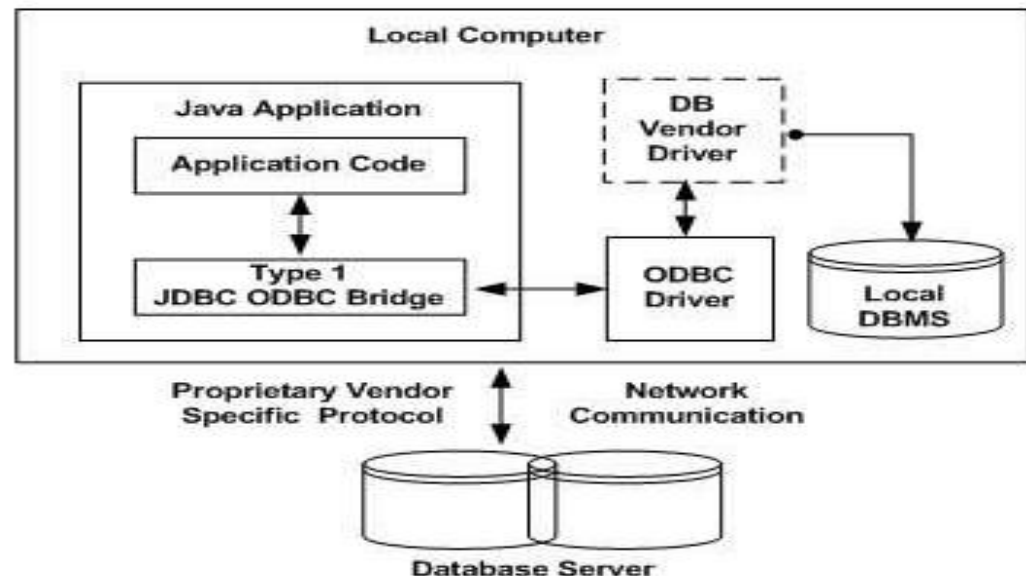


JDBC ARCHITECTURE/JDBC Driver Types

- **Type 1**
 - JDBC-ODBC Bridge
 - **Type 2**
 - Native API, partially java driver
 - **Type 3**
 - JDBC Network Driver, partially java
 - **Type 4**
 - Native-protocol pure Java driver (100% Java)
-

JDBC ARCHITECTURE/JDBC Driver Types

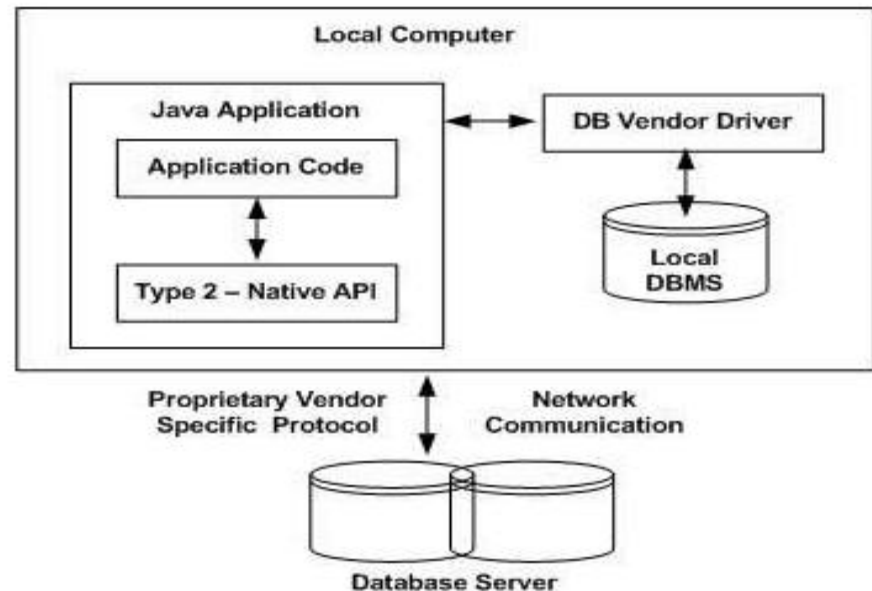
- **Type 1 Driver:** JDBC-ODBC Bridge Driver
- Translate JDBC into ODBC and use Windows ODBC built in drivers
- ODBC must be set up on every client
 - driver must be physically on each machine for both java applications and applets
 - for server side applications, ODBC must be set up on web server
- driver `oracle.jdbc.odbcc.JdbcOdbc` provided by JavaSoft with JDK



Eg: JDBC-ODBC Bridge that comes with JDK 1.2

JDBC ARCHITECTURE/JDBC Driver Types

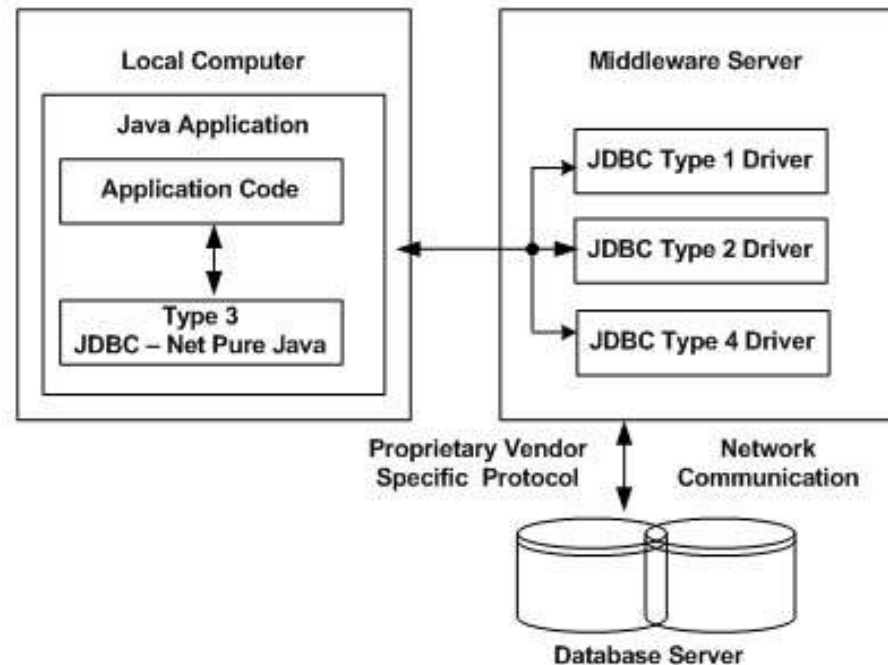
- **Type 2 Driver: JDBC-Native API**
- Native-API partly-Java driver converts JDBC calls into calls on the client API for Oracle, Sybase, Informix or other DBMS
- like Type 1 drivers; requires installation of binaries on each client & not suitable for large networks.



Eg: Oracle Call Interface (OCI) driver.

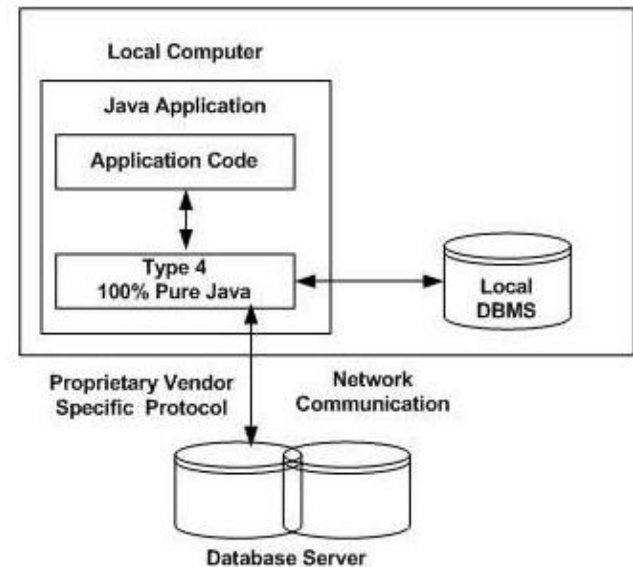
JDBC ARCHITECTURE/JDBC Driver Types

- **Type 3 Driver:** JDBC-Net pure Java
- three-tier approach is used to access databases. The JDBC clients use standard network sockets to communicate with a middleware application server.
- The middleware connects its pure Java clients to many different databases. The type of protocol in this middleware depends on the vendor.
- Most flexible driver type



JDBC ARCHITECTURE/JDBC Driver Types

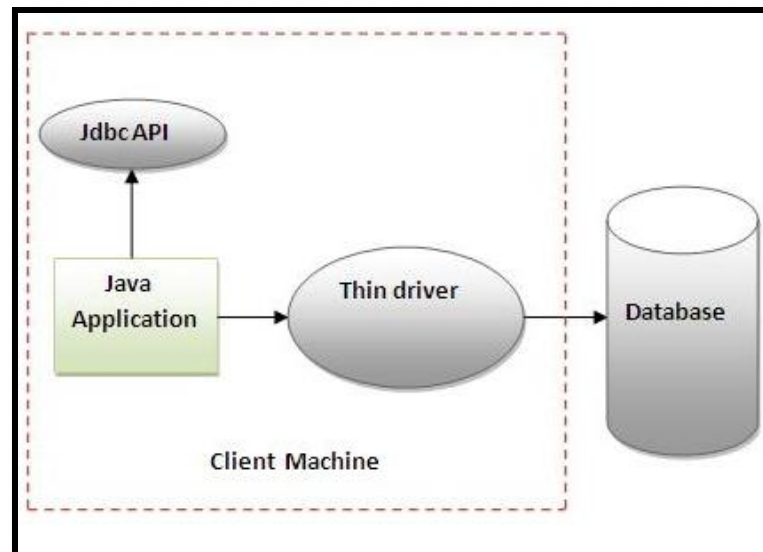
- **Type 4:** 100% Pure Java
- Converts JDBC directly to native API used by the RDBMS
- Requests from client machines are made directly to the DBMS server.
- Compiles into the application, applet & does not require anything to be installed on client machine, except JVM
- They are also written in 100% Java and are the most efficient among all driver types.



Eg: MySQL's Connector/J driver is a Type 4 driver.

JDBC ARCHITECTURE/JDBC Driver Types

- A **Type 4 driver** is a pure Java driver, which usually comes as a .jar file and performs direct calls to the database server. It does not need any configuration on the client's machine, other than including the name of the main driver's class in your Java code. That's why it's also known as the *thin driver*.
- For example, Java applets can be packaged with this type of driver, which can be automatically downloaded to the user's machine along with the applets themselves.



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.
-

Working with Databases using JDBC

- A connection can be open with the help of following steps:
 1. **Importing Packages**
 2. **Loading JDBC Drivers**
 3. **Opening a Connection to a Database**
 4. **Creating a Statement Object**
 5. **Executing a Query and Returning a Result Set Object**
 6. **Processing the Result Set**
 7. **Closing the Result Set and Statement Objects**
 8. **Closing the Connection**
-

- Driver interface
 - Connection interface
 - Statement interface
 - PreparedStatement interface
 - CallableStatement interface
 - ResultSet interface
 - ResultSetMetaData interface
 - DatabaseMetaData interface
 - RowSet interface
-

Working with Databases using JDBC

- **Importing Packages**

- `import java.sql.* ;` // for standard JDBC programs
- `import java.math.* ;` // for Decimal and Integer support
- JDBC API is in the package `java.sql`

It consists of interfaces, classes and exceptions .

Interfaces:

- CallableStatement, **Connection**, DatabaseMetaData, **Statement**, Driver, PreparedStatement, **ResultSet**, ResultSetMetaData etc.

Classes:

- Date, **DriverManager**, DriverPropertyInfo, Time, Timestamp, Types

Exceptions:

- DataTruncation, SQLException, SQLWarning
-

Working with Databases using JDBC

- **Loading JDBC Drivers**

- Load the JDBC driver using the method `forName()` of the Java class `Class`.
- If you work with Oracle DBMS, load a Type 4 JDBC driver with the following command: **`Class.forName("oracle.jdbc.driver.OracleDriver");`**

The return type of the `Class.forName (String ClassName)` method is "Class".
Class is a class in `java.lang` package.

JDBC DRIVER	
Data base	Driver class
Access	<code>sun.jdbc.odbc.JdbcOdbcDriver</code>
MySql	<code>com.mysql.jdbc.Driver</code>
Oracle	<code>oracle.jdbc.driver.OracleDriver</code>

Working with Databases using JDBC

- **Opening a Connection to a Database**

- The **JDBC DriverManager** class defines objects which can connect Java applications to a JDBC driver. DriverManager class manages the JDBC drivers that are installed on the system.
- Its **getConnection()** method is used to establish a connection to a database. It uses a username, password, and a jdbc url to establish a connection to the database and returns a **connection object**.
- Obtain the database connection to the database by calling
DriverManager.getConnection(url, user, password);

```
Connection con=  
DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","oracle");
```

- **Driver:** thin, If the database is on the same computer, then for the Host Name parameter, enter **localhost**, **JDBC Port:** 1521, **SID:** xe

Working with Databases using JDBC

- Following table lists down the popular JDBC driver names and database URL.

RDBMS	JDBC driver name	URL format
ORACLE	oracle.jdbc.driver.OracleDriver	jdbc:oracle:thin: @hostname:portno:databaseName
MySQL	com.mysql.jdbc.Driver	jdbc:mysql: //hostname/ databaseName
DB2	COM.ibm.db2.jdbc.net.DB2Driver	jdbc:db2: hostname:port Number/databaseName
Sybase	com.sybase.jdbc.SybDriver	jdbc:sybase:Tds: hostname: portno/databaseName

All the highlighted part in URL format is static and you need to change only the remaining part as per your database setup.

Working with Databases using JDBC

JDBC URLs Examples	
Database	Examples
Access	<code>Connection con = DriverManager.getConnenction("jdbc:odbc:ExampleMDBDataSource");</code>
MySql	<code>Connection con = DriverManager.getConnenction("jdbc:mysql://localhost/javabook","scott","tiger");</code>
Oracle	<code>Connection con= DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","oracle");</code>

Working with Databases using JDBC

- **Creating a Statement Object**

- Once a connection is established, It is used to pass SQL statements to its underlying database.
 - JDBC provides three classes for sending SQL Statements to the database, where PreparedStatement extends from Statement and CallableStatement extends from PreparedStatement
 - Statement :
For simple SQL statements (no parameter)
 - PreparedStatement :
For SQL statements with one or more IN parameters, or simple SQL statements that are executed frequently.
 - CallableStatement :
For executing SQL stored procedures.
-

Working with Databases using JDBC

- **Creating a Statement Object**

- Create an instance of the Java class Statement:

`Connection.createStatement();`

- When an SQL statement is to be issued against the database, a Statement object must be created through the Connection. To execute SQL statements, you need to instantiate a Statement object from your connection object by using the `createStatement()` method. A statement object is used to send and execute SQL statements to a database.

`Statement stmt= con.createStatement();`

where con is connection object.

Working with Databases using JDBC

- **Executing a Query and Returning a Result Set Object**
- A Statement is an interface that represents a SQL statement. You execute Statement objects, and they generate ResultSet objects, which is a table of data representing a database result set.
 - **ResultSet rs=stmt.executeQuery("select * from emp");**
- The statement interface provides three different methods for executing SQL statements :

-SELECT	executeQuery(String sql)
-INSERT, UPDATE, DELETE, CREATE, DROP	executeUpdate(String sql)
-Stored procedure	execute(String sql)

Working with Databases using JDBC

- **Processing the Result Set**

- Write a loop to process the database result set, if any:

- `while (rs.next())`

```
{
```

```
//Lines of Code.....
```

```
}
```

where rs is ResultSet object.

- `rs.next()` returns true if the new current row is valid, false if there are no more rows.
 - The other methods which can be used with Resultset object are :
 - `first()`,`last()`,`previous()`
-

Working with Databases using JDBC

- **Closing the Result Set ,Statement Objects & Closing the Connection**
 - Free system resources by closing the ResultSet, Statement and Connection objects.
 - Resultset
 - Statement/Prepared Statement
 - Connections
 - Closing the connections using close() method.
 - At the end of your JDBC program, it is required explicitly to close all the connections to the database to end each database session.
 - However, if you forget, Java's garbage collector will close the connection when it cleans up stale objects.
-

JDBC Examples

- **Retrieve the data from oracle by using JDBC.**
- Install Oracle and make sure it is running (ie. connect to Oracle by sqlplus).
- Create a table name **student**:

Column name	Data Type
rollno	varchar2(10)
name	char(25)
date_adm	date
per	number(4,2)

- Insert data into **student** table.
 - commit the operation.
-

JDBC Examples

```
SQL> select * from student;
```

ROLLNO	NAME	DATE_ADM	PER
13BCS1021	Ajay Sharma	18-JUL-13	80.54
13BCS1030	Amrinder Singh	11-JUL-13	78.2
13BCS1045	Arvind	25-JUN-13	75
13BCS1015	Aditya Sharma	05-AUG-13	76.56
13BCS1085	Gurpreet Kaur	20-JUN-13	85.76
14BCS8002	Aishwarya	10-AUG-14	81.45

```
6 rows selected.
```

```
SQL> desc student;
```

Name	Null?	Type
ROLLNO		VARCHAR2(10)
NAME		CHAR(25)
DATE_ADM		DATE
PER		NUMBER(4,2)

```
SQL> commit;
```

```
Commit complete.
```

JDBC Examples

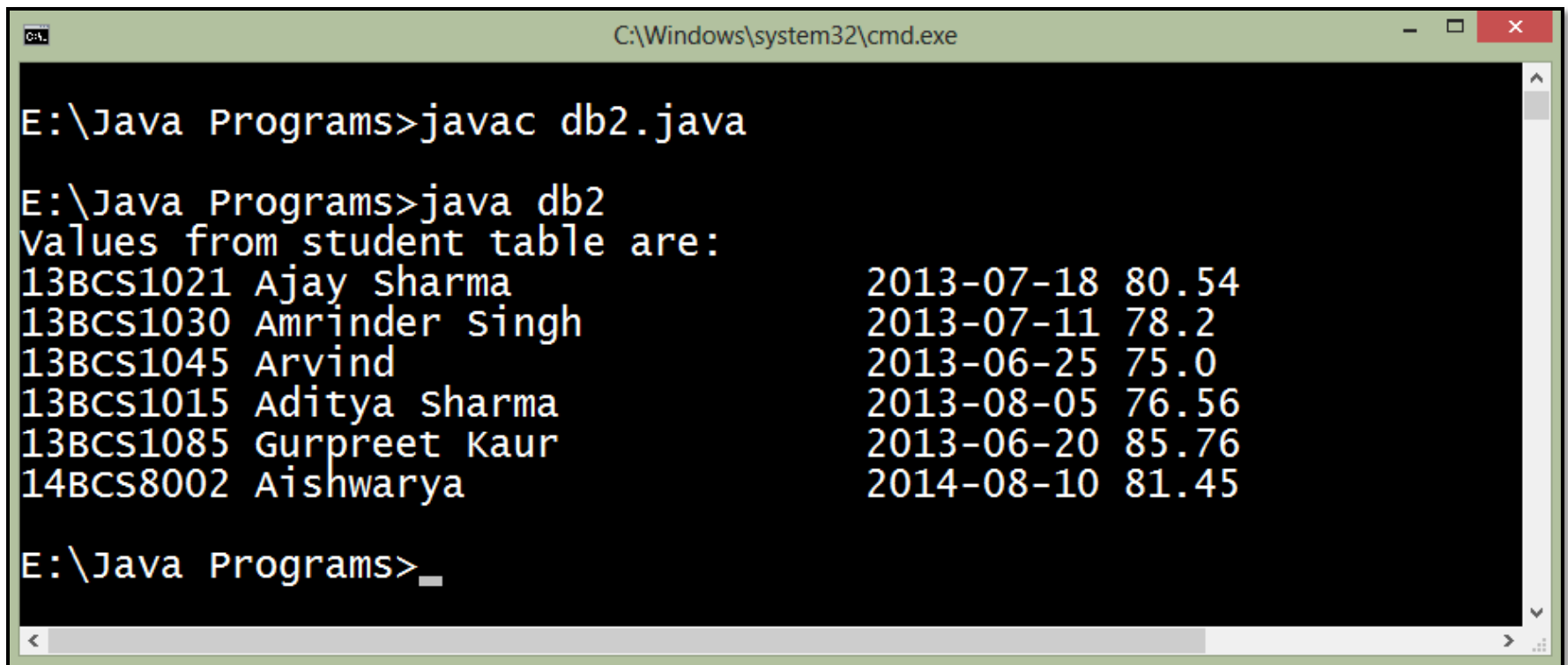
```
import java.sql.*;
class db2
{
    public static void main(String args[])
    {
        try
        {
            //step1 load the driver class
            Class.forName("oracle.jdbc.driver.OracleDriver");
            //step2 create the connection object
            Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","oracle");
            //step3 create the statement object
            Statement stmt=con.createStatement();
            //step4 execute query
            ResultSet rs=stmt.executeQuery("select * from student");
            System.out.println("Values from student table are:");
```

JDBC Examples

```
while(rs.next( ))
    System.out.println(rs.getString(1)+" "+rs.getString(2)+" "+rs.getDate(3)+" "+rs.getFloat(4));
    //step5 close the connection object
con.close( );
}
catch(Exception e)
{
    System.out.println(e);
}
}
```

JDBC Examples

OUTPUT



```
C:\Windows\system32\cmd.exe

E:\Java Programs>javac db2.java

E:\Java Programs>java db2
Values from student table are:
13BCS1021 Ajay Sharma          2013-07-18 80.54
13BCS1030 Amrinder Singh      2013-07-11 78.2
13BCS1045 Arvind              2013-06-25 75.0
13BCS1015 Aditya Sharma        2013-08-05 76.56
13BCS1085 Gurpreet Kaur        2013-06-20 85.76
14BCS8002 Aishwarya            2014-08-10 81.45

E:\Java Programs>
```

JDBC Examples

JDBC - Insert Records

//STEP 4: Execute a query

```
System.out.println("Inserting records into the table...");  
stmt = conn.createStatement();  
String sql = "INSERT INTO Student " + "VALUES (100, 'Zara', '10-jul-14',  
    80)";  
stmt.executeUpdate(sql);  
System.out.println("Inserted records into the table...");
```

JDBC Examples

Drop an existing Database::

//STEP 4: Execute a query

```
System.out.println("Deleting database...");  
stmt = conn.createStatement();  
String sql = "DROP DATABASE STUDENTS";  
stmt.executeUpdate(sql);  
System.out.println("Database deleted successfully...");
```

JDBC Examples

Create a table:

//STEP 4: Execute a query

```
System.out.println("Creating table in given database...");  
stmt = conn.createStatement();
```

```
String sql = "CREATE TABLE REGISTRATION " + "(id INTEGER not  
NULL, " + " first VARCHAR(255), " + " last VARCHAR(255), " + "  
age INTEGER, " + " PRIMARY KEY ( id ))";  
stmt.executeUpdate(sql);  
System.out.println("Created table in given database...");
```

JDBC Examples

JDBC - Drop Tables

//STEP 4: Execute a query

```
System.out.println("Deleting table in given database...");  
stmt = conn.createStatement();  
String sql = "DROP TABLE REGISTRATION ";  
stmt.executeUpdate(sql);  
System.out.println("Table deleted in given database...");
```
