



JDBC

Introduction

- Java application cannot directly communicate with database
- This is because a database can interpret only SQL statements and not java language statements for this reason, we need a mechanism to translate java statements into SQL statements
- JDBC API provides the mechanism for the kind of translation
- JDBC stands for Java Database Connectivity
- It is a standard Java API for database-independent connectivity between the Java programming language and a wide range of databases

Why JDBC API

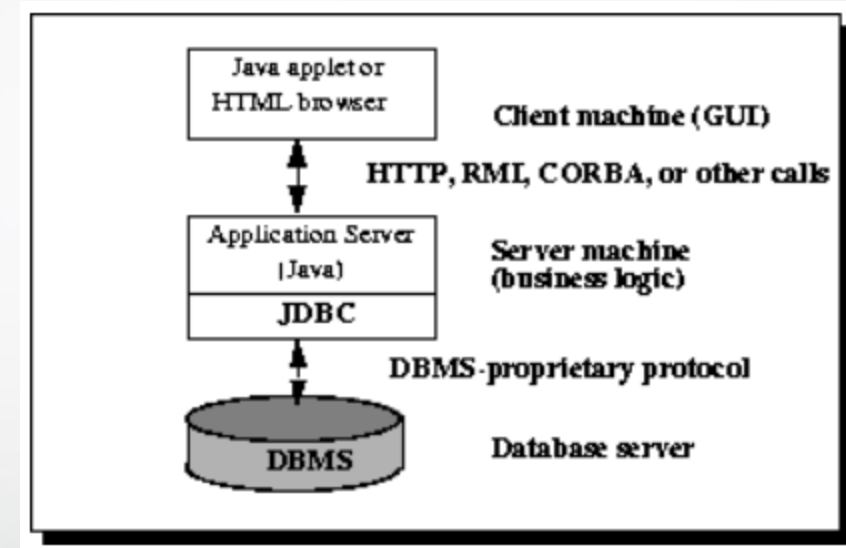
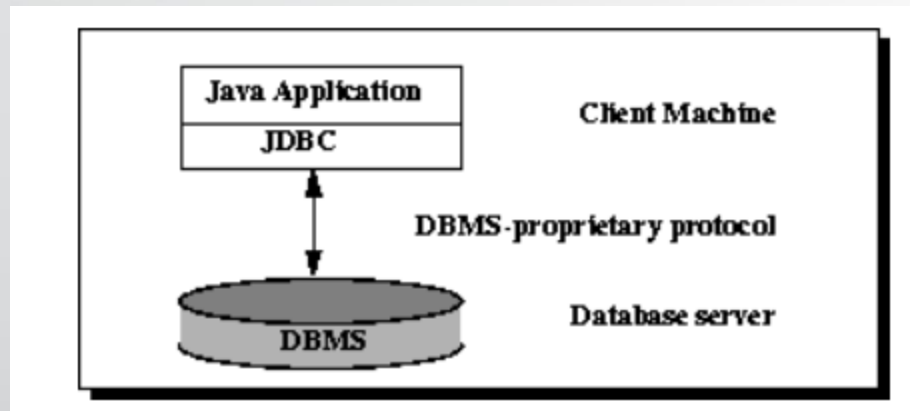
- Two issues :
 - 1) Java applications cannot directly communicate with a database.
 - 2) Java application should be database independent.

JDBC API uses driver to address the above issues.

How?

- JDBC API takes care of converting java Commands to generic SQL statements.
- JDBC API uses drivers provided by Database vendors to communicate with database.
- Java Application invokes methods JDBC API.

JDBC



Categories of JDBC drivers

There are several categories of JDBC drivers provided by different database vendors:

1) JDBC-ODBC driver:-

- Database server contain the ODBC driver embedded into them.
- ODBC APIs are written in C language and makes the use of pointers and other constructs that JAVA doesnot support.
- Hence JDBC-ODBC bridge driver are used to translate the JDBC API to the ODBC API.

2) Native API partly Java Driver:

Supplied by vendors of database like DB2, Informix provides JDBC drivers in terms of classes that are directly invoked by JDBC API.

3) Native protocol pure java driver/JDBC-Net pure java driver:

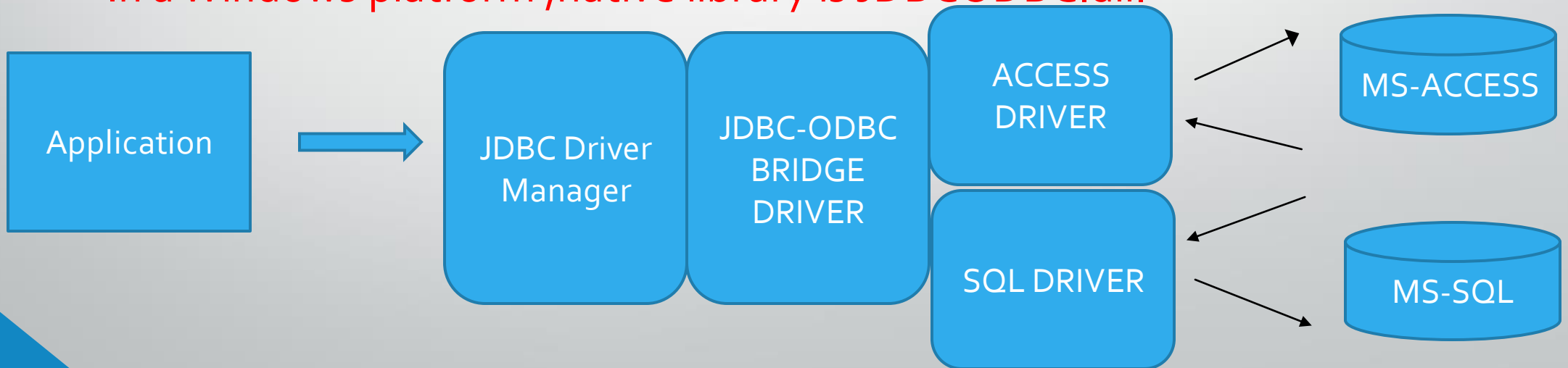
These drivers are used to connect a client application or applet to a database over a TCP/IP connection.

JDBC Driver Manager

- It is a backbone of the JDBC architecture.
- The function of **JDBC driver manager** are to maintain a list of drivers created for different databases and
- Connect a Java application to appropriate driver specified in a Java program.

JDBC-ODBC bridge:

- It is implemented as the **JdbcOdbc.class** and a **native library is used to access the ODBC driver.**
- **In a Windows platform ,native library is JDBCODBC.dll.**



JDBC Application Architecture Using the JDBC-ODBC Bridge Driver

Steps to connect to the database

There are 5 steps to connect any java application with the database using JDBC.

These steps are as follows:

1. Register the Driver class
2. Create connection
3. Create statement
4. Execute queries
5. Close connection

1. Register the driver class

- `forName` method of `Class` class is used to register the driver class
- This method is used to dynamically load the driver class

Syntax of `forName` method:

```
public static void forName (String className )throws ClassNotFoundException
```

Calling a method:

```
Class.forName("com.mysql.cj.jdbc.Driver");
```


2. Create the connection object

`getConnection` method of `DriverManager` class is used to establish connection with the database.

Syntax of `getConnection()` method: (two forms)

- `public static Connection getConnection(String url)throws SQLException`
- `public static Connection getConnection(String url,String name,String password)throws SQLException`

Calling a method:

Connection con=

```
DriverManager.getConnection("jdbc:mysql://localhost:3306/Student","root","root");
```

3. Create the Statement object

- `createStatement()` method of Connection interface is used to create statement.
- object of statement is responsible to execute queries with the database.

Syntax of `createStatement()` method:

```
public Statement createStatement()throws SQLException
```

Calling a method:

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

4. Execute the query

- `executeQuery()` method of Statement interface is used to execute queries to the database.
- This method returns the `object of ResultSet` that can be used to get all the records of a table

Syntax of `executeQuery()` method

```
public ResultSet executeQuery (String sql )throws SQLException
```

Calling a method:

```
ResultSet rs= stmt.executeQuery ("select * from stud");
```

4. Execute the query Contd..

- **ResultSet object** provides you with methods to access data from a table.
 - 1) It maintains a cursor pointing to its current row of data
 - 2) Initially the cursor is positioned before the first row.
 - 3) The next() moves the cursor to the next row.
 - 4) We can retrieve data from ResultSet rows by calling getXXX(int cn)
 - 5) XXX refers to a database of a column such as String ,Integer and Float and cn specifies Column number in the ResultSet.

SQL Type	Java class	ResultSet get method
BIT	Boolean	getBoolean()
CHAR	String	getString()
VARCHAR	String	getString()
DOUBLE	Double	getDouble()
FLOAT	Double	getDouble()
INTEGER	Integer	getInt()
REAL	Double	getFloat()
DATE	java.sql.Date	getDate()
TIME	java.sql.Time	getTime()
TIMESTAMP	java.sql.TimeStamp	getTimestamp()

How to execute parametrized query

- PreparedStatement object allows you to execute the parametrized queries.
- Let's see the example of parameterized query:
- `String sql="insert into emp values(?,?,?)";`

Why use PreparedStatement?

- As you can see, we are passing parameter (?) for the values. Its value will be set by calling the setter methods of PreparedStatement.
- Improves performance: The performance of the application will be faster if you use PreparedStatement interface because query is compiled only once.

How to execute parametrized query

- Ex:

```
PreparedStatement stmt=con.prepareStatement("insert into Emp values(?,?)");  
stmt.setInt(1,101);//1 specifies the first parameter in the query  
stmt.setString(2,"Ratan");
```

```
int i=stmt.executeUpdate();  
System.out.println(i+" records inserted");
```

5. Close the connection object


- By closing connection object, Statement and ResultSet will be closed automatically
- The `close()` method of Connection interface is used to close the connection

Syntax of `close()` method:

```
public void close() throws SQLException
```

Calling a method:

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

JDBC program to insert record
in a stud table of database
Student

```
Import java.sql.*;

public class JDBCDEMO
{ // JDBC driver name and database URL

static final String JDBC_DRIVER = "com.mysql.cj.jdbc.Driver";

static final String DB_URL =
"jdbc:mysql://localhost:3306/Student";

// Database credentials

static final String USER = "root";
static final String PASS = "root";

public static void main(String[] args) {

Connection conn = null;
Statement stmt = null;

try{

//Register JDBC driver

Class.forName(JDBC_DRIVER);

//Open a connection

System.out.println("Connecting to database...");
conn = DriverManager.getConnection(DB_URL,USER,PASS);
```

//Execute a query

```
System.out.println("Creating statement...");

stmt = conn.createStatement();

String sql;

sql = "insert into stud(name,rollno) values ('xyz','3')";

int r= stmt.executeUpdate(sql);

ResultSet rs = stmt.executeQuery("Select * from stud");

//Extract data from result set

while(rs.next()){

System.out.println("Roll no: "+rs.getInt(2)+" Name:
"+rs.getString(1));      }

//Clean-up environment

rs.close();  stmt.close();  conn.close();

}catch(SQLException se){

        se.printStackTrace();  }

catch(Exception e){

e.printStackTrace();  }

System.out.println("Goodbye!");

}
```

Output

Command Prompt

```
C:\Users\juhig\OneDrive\OOPM\Programs\Module6>javac JDBCDEMO.java
```

```
C:\Users\juhig\OneDrive\OOPM\Programs\Module6>java -cp .;"C:\Program Files\Java\jdk-14.0.2\lib\ext1\mysql-connector-java-8.0.22.jar" JDBCDEMO
```

```
Connecting to database...
```

```
Creating statement...
```

```
Roll no: 3 Name: xyz
```

```
Goodbye!
```

```
C:\Users\juhig\OneDrive\OOPM\Programs\Module6>_
```