JDBC –
Java DataBase Connectivity

Satish Bansal

What is JDBC?

- JDBC is a Java-based data access technology (Java Standard Edition platform) from Oracle Corporation.
- This technology is an API for the Java programming language that defines how a client may access a database in database independent and platform independent manner.
- It provides classes and interfaces to connect or communicate Java application with database. It provides methods for querying and updating data in a database.

Why JDBC

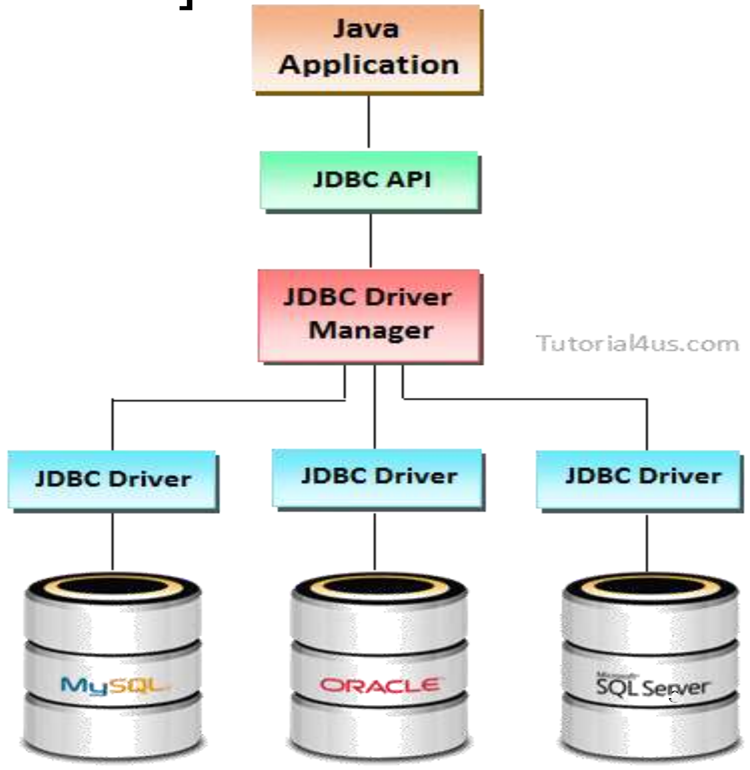
- Before JDBC, ODBC API was used to connect and execute query to the database.
- But ODBC API uses ODBC driver that is written in C language which is platform dependent and unsecured.
- That is why Sun Micro System has defined its own API (JDBC API) that uses JDBC driver written in Java language.

Importance of JDBC

- Java application is platform independent but if it is combined with ODBC then it become platform dependent but this is against of java motto or principal. To solved the above problems, Sum MicroSystem introduced JDBC technology.

	ODBC	JDBC
1	Odbc is platform dependent and database independent.	Jdbc is both platform and database independent.
2	Odbc implemented in C language with pointer.	Jdbc implemented in java without pointer.

- Jdbc ia a part of JDK software so no need to install separate software for Jdbc API
- Jdbc API consists of two packages
- java.sql package[This package include classes and interface to perform almost all JDBC operation such as creating and executing SQL Queries.]
- javax.sql package[This package is also known as JDBC extension API. It provides classes and interface to access server-side data.]



- **Jdbc API**
- **Packages**
- java.sql package
- javax.sql package
- **Classes In Jdbc**
- DriverManager
- SQLException
- Types
- Date
- Time
- **Interfaces In Jdbc**
- Connection
- Statement
- PreparedStatement
- CallableStatementResultset
- ResultSetMetaData
- DatabaseMetaData
- Driver

DSN (Data Source Name)

- DSN (Data Source Name) is a file which stores a database name. An ODBC driver uses a DSN file and reads a database name then connects with that database.
- **Tpyes of DSN**
- **System DSN:** System DSN will be stored in a shareable location of registry and this type of DSN is accessible for multiple user accounts of the system.
- **User DSN:** User DSN will be stored in a non-shareable location of registry and this type of DSN is only accessible for one user account of the system.
- **File DSN:** File DSN is like user DSN, but the DSN will be stored in a user given location of hard disk.

JDBC Drivers

- JDBC Driver is a software component that enables java application to interact with the database.

There are 4 types of JDBC drivers: –

Type 1: JDBC-ODBC bridge driver –

Type 2: Native-API driver (partially java driver) –

Type 3: Network Protocol driver (fully java driver)

Type 4: Thin driver (fully java driver)

Short Description of Jdbc drivers

Driver	Database	Platform
Type - 1	Independent	Dependent.
Type - 2	Dependent	Dependent.
Type - 3	Independent	Independent.
Type - 4	Dependent	Independent.

Type-1:JDBC Odbc Bridge Driver

- Type-1:The JDBC-ODBC bridge driver uses ODBC driver to connect to the database. This is now discouraged because of thin driver. It is install automatically along with JDK software. This driver convert JDBC calls into Odbc calls(function) So this is called a **bridge driver**.
- **Advantage of bridge driver**
- Easy to use
- It is a database independent driver
- Can be easily connected to any database.
- This driver software is built-in with JDK so no need to install separately.
- **Disadvantage of bridge driver**
- It is a slow driver so not used in real time application
- Because of Odbc connectivity it is a platform dependent driver.
- It is not a portable driver.
- It is not suitable for applet to connect with database.

Type 2:Native API Driver

- Type-2:The Native API driver uses the client-side libraries of the database.
- It is not written entirely in Java.
- Native API driver uses native API to connect a java program directly to the database. Native API id's C, C++ library, which contains a set of function used to connect with database directly. Native API will be different from one database to another database. So this Native API driver is a database dependent driver.
- **Advantage of Thin driver**
- Native API driver comparatively faster than JDBC-ODBC bridge driver.
- **Disadvantage of Thin driver**
- Native API driver is database dependent and also platform dependent because of Native API.

Type 3:Network Protocol Driver

- Type-3:The Network Protocol driver uses middle ware (application server).
- It is fully written in Java.
- **Advantage of Network Protocol driver**
- No client side library is required because of application server that can perform many tasks like auditing, load balancing, logging etc.
- This driver is both database and platform independent driver
- **Disadvantage of Network Protocol driver**
- Network support is required on client machine.
- Requires database-specific coding to be done in the middle tier.
- Maintenance of Network Protocol driver becomes costly because it requires database-specific coding to be done in the middle tier.

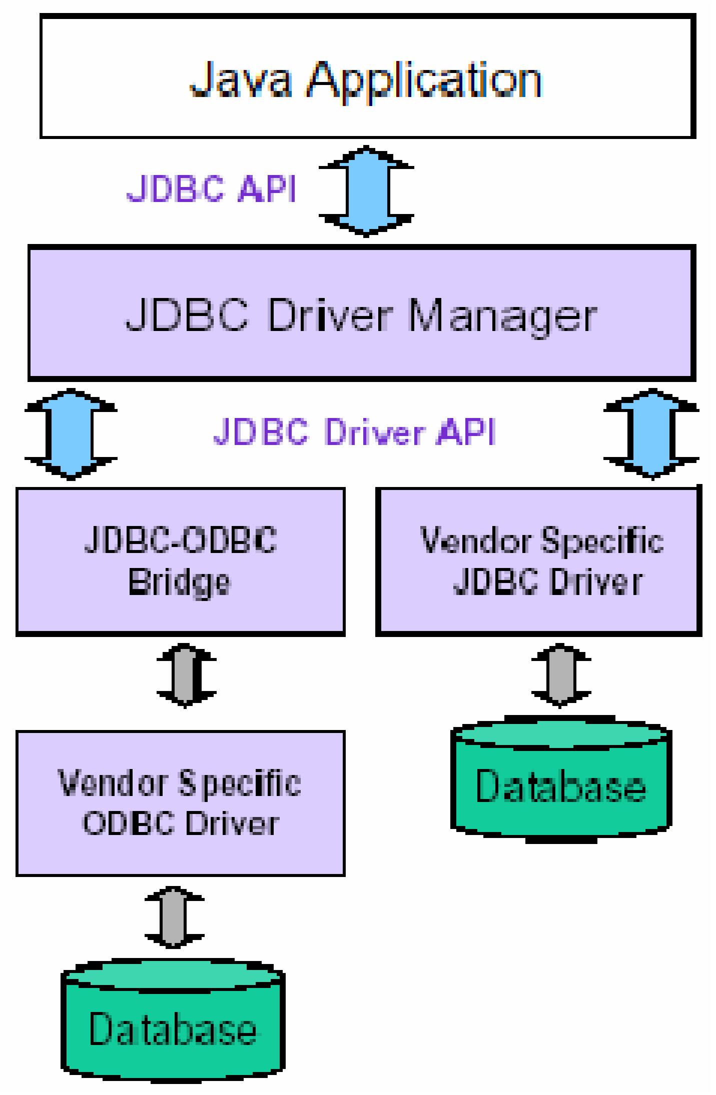
Type 4:Thin Driver

- Type-4:The thin driver converts JDBC calls directly into the vendor-specific database protocol.
- That is why it is known as thin driver. ● It is fully written in Java language.
- This thin driver uses the following three information to connect with a database.

1. Ip address of a machine (system), where the database server is running.
2. Port number of the database server.
3. Database name, also called SID (service ID).

- Thin driver is the fastest driver among all Jdbc drivers.
- Better performance than all other drivers.
- It can be used to connect an applet with the database.
- No software is required at client side or server side.
- Disadvantage: Drivers depends on the Database.

General Architecture



Connection Interface in JDBC

- A Connection is the session between java application and database. The Connection interface is a factory of Statement, PreparedStatement, and DatabaseMetaData i.e. object of Connection can be used to get the object of Statement and DatabaseMetaData.
- **Method of Connection Interface**

	method	Discription
1	public Statement createStatement()	creates a statement object that can be used to execute SQL queries.
2	public void commit()	saves the changes made since the previous commit/rollback permanent.
3	public void rollback()	Drops all changes made since the previous commit/rollback.
4	public void close():	closes the connection and Releases a JDBC resources immediately.

Drivermanager Class of JDBC

- The DriverManager class acts as an interface between user and drivers.
- **Methods of DriverManager Class**

	method	Description
1	public static void registerDriver(Driver driver)	Used to register the given driver with DriverManager.
2	public static void deregisterDriver(Driver driver):	Used to registered the given driver (drop the driver from the list) with DriverManager.
3	public static Connection getConnection(String url):	Used to establish the connection with the specified url.
4	public static Connection getConnection(String url,String userName,String password):	Used to establish the connection with the specified url, username and password.

Statement Interface in JDBC

- The Statement interface provides methods to execute queries with the database.

	method	Description
1	public ResultSet executeQuery(String sql)	Used to execute SELECT query. It returns the object of ResultSet.
2	public int executeUpdate(String sql)	Used to execute specified query, it may be create, drop, insert, update, delete etc.
3	public boolean execute(String sql)	Used to execute queries that may return multiple results.

ResultSet Interface in JDBC

- The object of ResultSet maintains a cursor pointing to a particular row of data. Initially, cursor points to before the first row.
- **Method of ResultSet Interface**

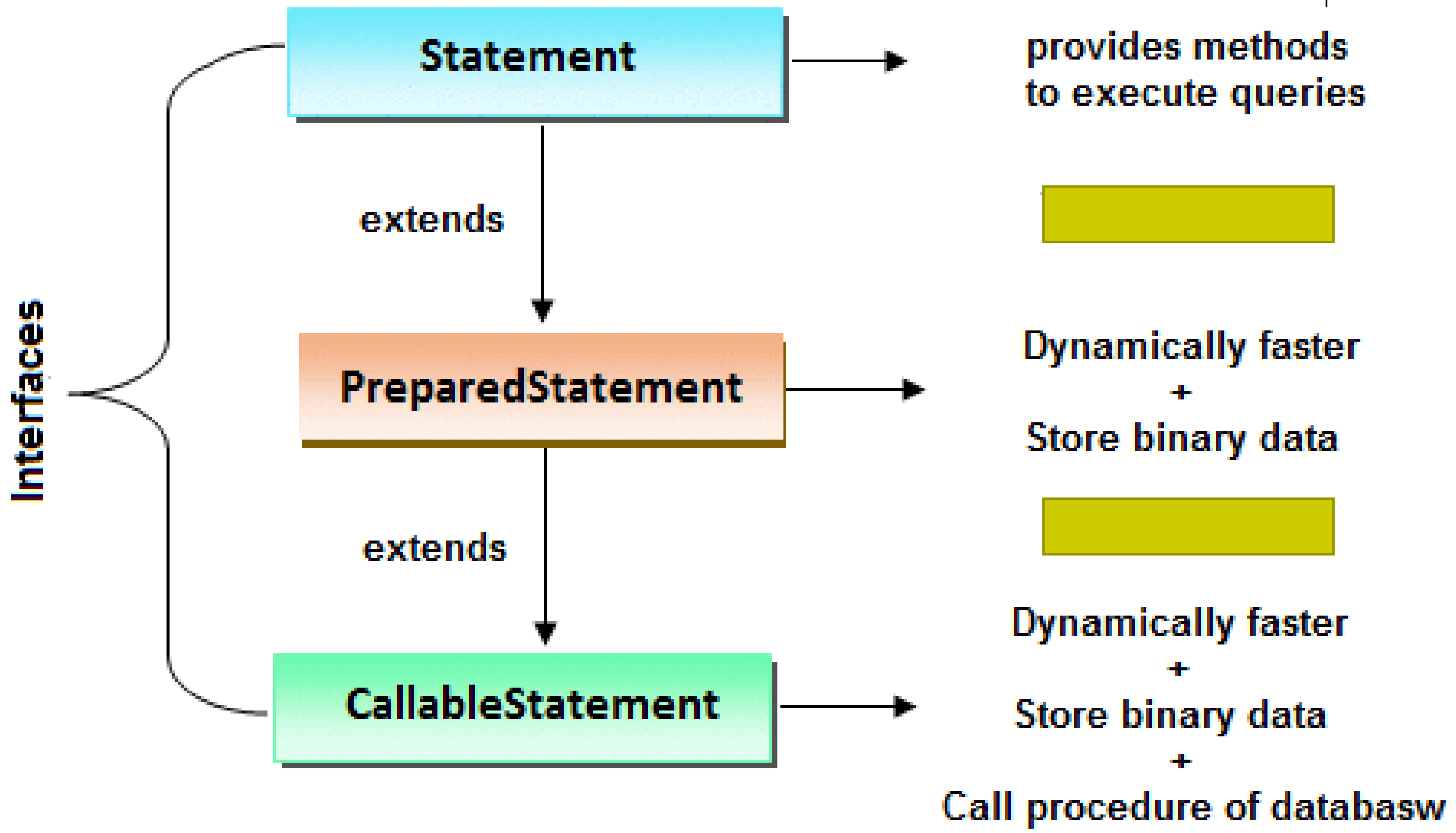
	method	Description
1	public boolean next()	Used to move the cursor to the one row next from the current position.
2	public boolean previous()	Used to move the cursor to the one row previous from the current position.
3	public boolean first()	Used to move the cursor to the first row in result set object.
4	public boolean last()	Used to move the cursor to the last row in result set object.

Preparedstatement Interfaces

- If the sql command is same then actually no need to compiling it for each time before it is executed. So the performance of an application will be Increased. In this case **PreparedStatement** is used.
- **Difference between PreparedStatement and Statement**

	Statement	PreparedStatement
1	Statement interface is slow because it compile the program for each execution	PreparedStatement interface is faster, because its compile the command for once.
2	We can not use ? symbol in sql command so setting dynamic value into the command is complex	We can use ? symbol in sql command, so setting dynamic value is simple.
3	We can not use statement for writing or reading binary data (picture)	We can use PreparedStatement for reading or writing binary data.

PreparedStatement Interface is derived Interface of statement and CallableStatement is derived Interface of PreparedStatement.


Interfaces
Statement
provides methods to execute queries
extends
PreparedStatement
Dynamically faster + Store binary data
extends
CallableStatement
Dynamically faster + Store binary data + Call procedure of databasw

Basic steps to use a database in Java

0. Register the driver class

1. Establish a **connection**

2. Create JDBC **Statements**

3. Execute **SQL** Statements

4. GET **ResultSet**

5. **Close** connections

Register the driver class

- **import java.sql.*;**

- **Load the vendor specific driver**
  - Class.forName(”oracle.jdbc.driver.OracleDriver”);

- The forName() method of Class class is used to register the driver class.
- Class.forName("com.mysql.jdbc.Driver");

Establish a connection

- Creating Connection Object
- The getConnection() method of DriverManager class is used to establish connection with the database.
- **Make the connection**
  - Connection con = DriverManager.getConnection(″jdbc:oracle:thin:@oracle-prod:1521:OPROD″, username, passwd);
- Connection con=DriverManager.getConnection("jdbc:mysql://localhost:3306","root","password");

Create JDBC statement(s)

- Creating Statement Object
- The createStatement() method of Connection interface is used to create statement. The object of statement is responsible to execute queries with the database.
- Statement stmt=con.createStatement();
- Creates a Statement object for sending SQL statements to the database.

Executing SQL Statements

- Execute Query
- The 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.
- ResultSet rs=stmt.executeQuery("select * from emp");
- while(rs.next()){ System.out.println(rs.getInt(1)+" "+rs.getString(2)); }
- String insertLehigh = ˝Insert into Lehigh values“ + ˝(123456789,abc,100)˝;

  stmt.**executeUpdate**(insertLehigh);

Get ResultSet

```
String queryLehigh = "select * from Lehigh";

ResultSet rs = Stmt.executeQuery(queryLehigh);
//What does this statement do?

while (rs.next()) {
   int ssn = rs.getInt("SSN");
   String name = rs.getString("NAME");
   int marks = rs.getInt("MARKS");
}
```

Close connection

- By closing connection object statement and ResultSet will be closed automatically. The close() method of Connection interface is used to close the connection.
- stmt.close();
- con.close();

Sample program for MS Access

```
import java.sql.*;
class Test  {
    public static void main(String[] args)  {
        try {
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver"); //dynamic loading of driver
            String filename = "c:/db1.mdb"; //Location of an Access database
            String database = "jdbc:odbc:Driver={Microsoft Access Driver (*.mdb)};DBQ=";
            database+= filename.trim() + ";DriverID=22;READONLY=true}"; //add on to end
            Connection con = DriverManager.getConnection( database ,"","");
            Statement s = con.createStatement();
            s.execute("create table TEST12345 ( firstcolumn integer )");
            s.execute("insert into TEST12345 values(1)");
            s.execute("select firstcolumn from TEST12345");
```

Sample program(cont)

```
        ResultSet rs = s.getResultSet();
        if (rs != null) // if rs == null, then there is no ResultSet to view
        while ( rs.next() ) // this will step through our data row-by-row
        {   /* the next line will get the first column in our current row's ResultSet
            as a String ( getString( columnNumber) ) and output it to the screen */
            System.out.println("Data from column_name: " + rs.getString(1) );
        }
        s.close(); // close Statement to let the database know we're done with it
        con.close(); //close connection
      }
    catch (Exception err) { System.out.println("ERROR: " + err);  }
  }
}
```

Connect to SQL Server Database

- Driver class: com.microsoft.sqlserver.jdbc.SQLServerDriver
- Connection URL: jdbc:sqlserver://localhost:1433;databaseName=stu Username: The username for the sql database is root.
- Password: Given by the user at the time of installing the sql database
- Example: Connection con=DriverManager.getConnection("jdbc:sqlserver://localhost:1433;databaseName=stu","root",”1234");

Loading the .jar

- Download the MS SQL jar file: sqljdbc4.jar
- Paste the sqljdbc4.jar in the lib folder of source directory.
- Set the classpath

DriverManager clas

- The DriverManager class acts as an interface between user and drivers.
- It keeps track of the drivers that are available and handles establishing a connection between a database and the appropriate driver.
- Connection con = null;
  con=DriverManager.getConnection("jdbc:sqlserver://localhost:1433;databaseName =stu","root",”1234");

Connection Interface

- A Connection is the session between Java application and database.
- The Connection interface is a factory of Statement and PreparedStatement.
- Object of Connection can be used to get the object of Statement and PreparedStatement.

Statement Interface

- The Statement interface provides methods to execute queries with the database.
- The statement interface is a factory of ResultSet.
- It provides factory method to get the object of ResultSet.

Statement Interface

- Methods of Statement interface:
- public ResultSet executeQuery(String sql): is used to execute SELECT query. It returns the object of ResultSet.
- public int executeUpdate(String sql): is used to execute specified query, it may be create, drop, insert, update, delete etc.
- public boolean execute(String sql): is used to execute queries that may return multiple results.

Statement Interface

- Example:
- Statement stmt=con.createStatement();
- int result=stmt.executeUpdate("delete from table where id=xy");
- System.out.println(result+" records affected");
- con.close();

ResultSet interface

- The object of ResultSet maintains a cursor pointing to a particular row of data.
- Initially, cursor points to before the first row.

ResultSet Interface

- Methods of ResultSet interface:
- public int getInt(int columnIndex): is used to return the data of specified column index of the current row as int.
- public int getInt(String columnName): columnName):is used to return the data of specified column name of the current row as int.
- public String getString(int columnIndex): is used to return the data of specified column index of the current row as String.
- public String getString(String columnName): is used to return the data of specified column name of the current row as String.

ResultSet Interface

- Example:
- ResultSet rs=stmt.executeQuery("select * from table");
- //getting the record of 3rd row
- rs.absolute(3);
- System.out.println(rs.getString(1)+" "+rs.getString(2)+" "+rs.getString(3));
- con.close();

PreparedStatement Interface

- The PreparedStatement interface is a sub interface of Statement.
- It is used to execute parameterized query.
- Example of parameterized query: –
- String sql="insert into emp values(?,?,?)";

PreparedStatement Interface

- Methods of PreparedStatement:
- public void setInt(int paramIndex, int value): sets the integer value to the given parameter index.
- public void setString(int paramIndex, String value): sets the String value to the given parameter index.
- public void setFloat(int paramIndex, float value): sets the float value to the given parameter index.

PreaparedStatement Interface

- Methods of PreparedStatement:
- public void setDouble(int paramIndex, double value): sets the double value to the given parameter index.
- public int executeUpdate(): executes the query. It is used for create, drop, insert, update, delete etc.
- public ResultSet executeQuery(): executes the select query. It returns an instance of ResultSet.

PreaparedStatement Interface

- Example:
- 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");

CallableStatement Interface

- A callablestatement object is used for calling the stored procedure from JDBC program.
- A callable statement can contain variables that you supply each times you execute the call.
- When the stored procedure returns, computed values are retrieved through the callablestatement object.

How to execute a callablestatement

- To get the instance of callablestatement :
- CallableStatement Cste=con.prepareCall(“{call fun()}”);
- To pass input parameters
- Cste.setXXX(index, value);
- Callablestatement should be execute as:
- Cste.execute();
- To get the output parameters
- Cste.getXXX(index);