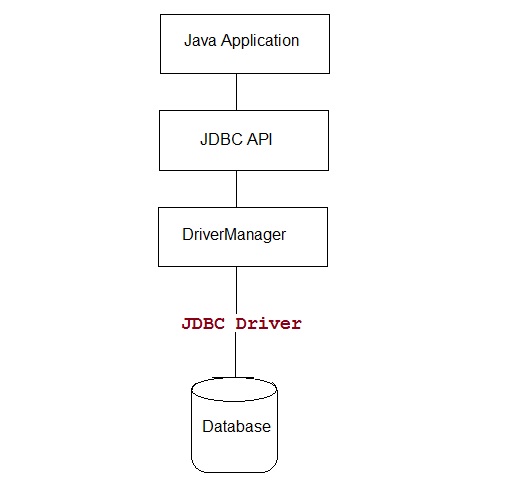
**Introduction to JDBC**

**References :** [**https://docs.oracle.com/javase/tutorial/jdbc/overview/index.html**](https://docs.oracle.com/javase/tutorial/jdbc/overview/index.html)

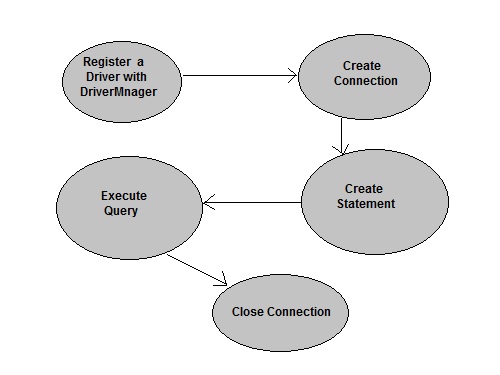
**Java Database Connectivity(JDBC)** is an **Application Programming Interface(API)** used to connect Java application with Database. JDBC is used to interact with various type of Database such as Oracle, MS Access, My SQL and SQL Server. JDBC can also be defined as the platform-independent interface between a relational database and Java programming. It allows java program to execute SQL statement and retrieve result from database.



**Steps to connect a Java Application to Database**

The following 5 steps are the basic steps involve in connecting a Java application with Database using JDBC.

1. Register the Driver
2. Create a Connection
3. Create SQL Statement
4. Execute SQL Statement
5. Closing the connection



**Register the Driver**

Class.forName() is used to load the driver class explicitly.

**Example to register with JDBC-ODBC Driver**

Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

**Create a Connection**

getConnection() method of **DriverManager** class is used to create a connection.

**Syntax**

getConnection(String url)

getConnection(String url, String username, String password)

getConnection(String url, Properties info)

**Example establish connection with Oracle Driver**

Connection con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE","username","password");

**Create SQL Statement**

createStatement() method is invoked on current **Connection** object to create a SQL Statement.

**Syntax**

public Statement createStatement() throws SQLException

**Example to create a SQL statement**

Statement s=con.createStatement();

**Execute SQL Statement**

executeQuery() method of **Statement** interface is used to execute SQL statements.

**Syntax**

public ResultSet executeQuery(String query) throws SQLException

**Example to execute a SQL statement**

ResultSet rs=s.executeQuery("select \* from user");

while(rs.next())

{

System.out.println(rs.getString(1)+" "+rs.getString(2));

}

**Closing the connection**

After executing SQL statement you need to close the connection and release the session. The close() method of **Connection** interface is used to close the connection.

**Syntax**

public void close() throws SQLException

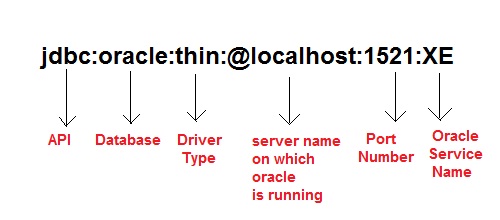
**Example of closing a connection**

con.close();

**Connecting to Oracle Database using Thin Driver**

To connect a Java application with Oracle database using Thin Driver. You need to follow the following steps

1. **Load Driver Class:** The Driver Class for oracle database is **oracle.jdbc.driver.OracleDriver** and Class.forName("oracle.jdbc.driver.OracleDriver") method is used to load the driver class for Oracle database.
2. **Create Connection:** For creating a connection you will need a Connection URL. The Connection URL for Oracle is



You will also require **Username** and **Password** of your Oracle Database Server for creating connection.

1. **Loading jar file:** To connect your java application with Oracle, you will also need to load ojdbc14.jar file. This file can be loaded into 2 ways.
   1. Copy the jar file into C:\Program Files\Java\jre7\lib\ext folder.

or,

* 1. Set it into classpath. For more detail see [**how to set classpath**](https://www.studytonight.com/java/setting-classpath-for-java.php)

[**Download ojdbc14.jar file**](https://www.studytonight.com/java/ojdbc.jar)

**NOTE:** Here we are discussing about Oracle 10g as database. For other version of Oracle you will be require to do some small changes in the Connection URL.

**Example**

**Create a table in Oracle Database**

create table Student(sid number(10),sname varchar2(20));

**Insert some record into the table**

insert into Student values(101,'adam');

insert into Student values(102,'abhi');

**Accessing record from Student table in Java application**

import java.sql.\*;

class Test

{

public static void main(String []args)

{

try{

//Loading driver

Class.forName("oracle.jdbc.driver.OracleDriver");

//creating connection

Connection con = DriverManager.getConnection

("jdbc:oracle:thin:@localhost:1521:XE","username","password");

Statement s=con.createStatement(); //creating statement

ResultSet rs=s.executeQuery("select \* from Student"); //executing statement

while(rs.next()){

System.out.println(rs.getInt(1)+" "+rs.getString(2));

}

con.close(); //closing connection

}catch(Exception e){

e.printStacktrace();

}

}

}

101 adam

102 abhi

**Inserting record into a table using java application**

import java.sql.\*;

class Test

{

public static void main(String []args)

{

try{

//Loading driver...

Class.forName("oracle.jdbc.driver.OracleDriver");

//creating connection...

Connection con = DriverManager.getConnection

("jdbc:oracle:thin:@localhost:1521:XE","username","password");

PreparedStatement pst=con.prepareStatement("insert into Student values(?,?)");

pst.setInt(1,104);

pst.setString(2,"Alex");

pst.executeUpdate();

con.close(); //closing connection

}catch(Exception e){

e.printStacktrace();

}

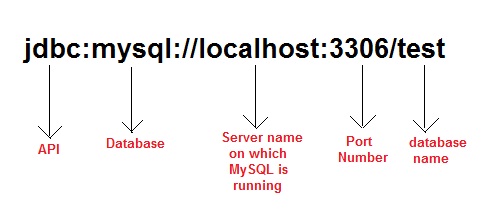
}

}

**Connecting to MySQL Database using Thin Driver**

To connect a Java application with MySQL database using Thin Driver. You need to follow the following steps

1. **Load Driver Class:** The Driver Class for MySQL database is **com.mysql.jdbc.Driver** and Class.forName("com.mysql.jdbc.Driver") method is used to load the driver class for MySQL database.
2. **Create Connection:** For creating a connection you will need a Connection URL. The Connection URL for MySQL is



You will also require **Username** and **Password** of your MySQL Database Server for creating connection.

1. **Loading jar file:** To connect your java application with MySQL, you will also need to load mysql-connector.jar file. This file can be loaded into 2 ways.
   1. Copy the jar file into C:\Program Files\Java\jre7\lib\ext folder.

or,

* 1. Set it into classpath. For more detail see [**how to set classpath**](https://www.studytonight.com/java/setting-classpath-for-java.php)

[**Download mysql-connector.jar file**](https://www.studytonight.com/java/mysql-connector.jar)

**Example**

**Create a table in MySQL Database**

create table Student(sid int(10),name varchar(20));

**Insert some record into the table**

insert into Student values(102,'adam');

insert into Student values(103,'abhi');

**Accessing record from Student table in Java application**

import java.sql.\*;

class Test

{

public static void main(String []args)

{

try{

//Loading driver

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

//creating connection

Connection con = DriverManager.getConnection

("jdbc:mysql:/ /localhost:3306/test","username","password");

Statement s = con.createStatement(); //creating statement

ResultSet rs = s.executeQuery("select \* from Student"); //executing statement

while(rs.next()){

System.out.println(rs.getInt(1)+" "+rs.getString(2));

}

con.close(); //closing connection

}catch(Exception e){

e.printStacktrace();

}

}

}

102 adam

103 abhi

**Inserting record into a table using java application**

import java.sql.\*;

class Test

{

public static void main(String []args)

{

try{

//Loading driver

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

//creating connection

Connection con = DriverManager.getConnection

("jdbc:mysql:/ /localhost:3306/test","username","password");

PreparedStatement pst=con.prepareStatement("insert into Student values(?,?)");

pst.setInt(1,104);

pst.setString(2,"Alex");

pst.executeUpdate();

con.close(); //closing connection

}catch(Exception e){

e.printStacktrace();

}

}

}

**What's new in JDBC 4.0**

**JDBC 4.0** is new and advance specification of JDBC. It provides the following advance features

* Connection Management
* Auto loading of Driver Interface.
* Better exception handling
* Support for large object
* Annotation in SQL query.

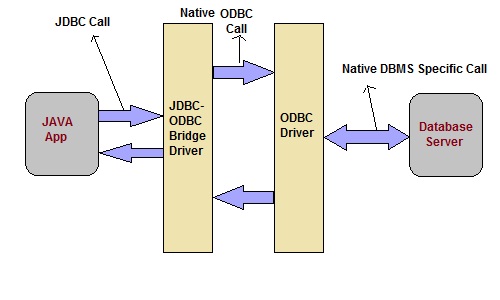
**JDBC Driver**

JDBC Driver is required to process SQL requests and generate result. The following are the different types of driver available in JDBC.

* **Type-1 Driver** or **JDBC-ODBC bridge**
* **Type-2 Driver** or **Native API Partly Java Driver**
* **Type-3 Driver** or **Network Protocol Driver**
* **Type-4 Driver** or **Thin Driver**

**JDBC-ODBC bridge**

**Type-1 Driver** act as a bridge between JDBC and other database connectivity mechanism(ODBC). This driver converts JDBC calls into ODBC calls and redirects the request to the ODBC driver.



**Advantage**

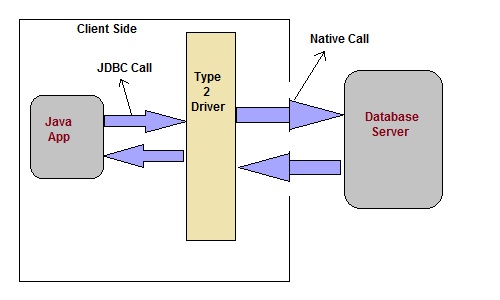
* Easy to use
* Allow easy connectivity to all database supported by the ODBC Driver.

**Disadvantage**

* Slow execution time
* Dependent on ODBC Driver.
* Uses Java Native Interface(JNI) to make ODBC call.

**Native API Driver**

This type of driver make use of Java Native Interface(JNI) call on database specific native client API. These native client API are usually written in C and C++.



**Advantage**

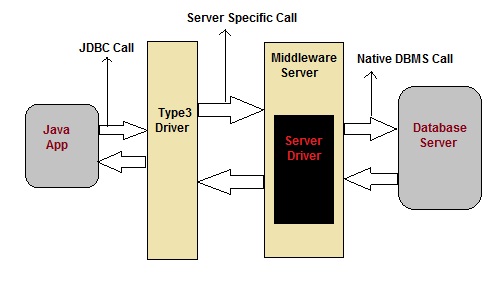
* faster as compared to **Type-1 Driver**
* Contains additional features.

**Disadvantage**

* Requires native library
* Increased cost of Application

**Network Protocol Driver**

This driver translate the JDBC calls into a database server independent and Middleware server-specific calls. Middleware server further translate JDBC calls into database specific calls.



**Advantage**

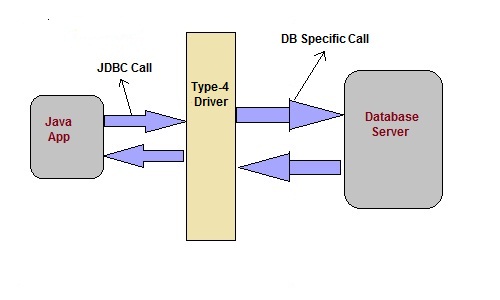
* Does not require any native library to be installed.
* Database Independency.
* Provide facility to switch over from one database to another database.

**Disadvantage**

* Slow due to increase number of network call.

**Thin Driver**

This is Driver called Pure Java Driver because. This driver interact directly with database. It does not require any native database library, that is why it is also known as Thin Driver.



**Advantage**

* Does not require any native library.
* Does not require any Middleware server.
* Better Performance than other driver.

**Disadvantage**

* Slow due to increase number of network call.

**JDBC 4.0 API**

JDBC 4.0 API is mainly divided into two package

1. java.sql
2. javax.sql

**java.sql package**

This package include classes and interface to perform almost all JDBC operation such as creating and executing SQL Queries.

**Important classes and interface of java.sql package**

|  |  |
| --- | --- |
| **classes/interface** | **Description** |
| java.sql.BLOB | Provide support for BLOB(Binary Large Object) SQL type. |
| java.sql.Connection | creates a connection with specific database |
| java.sql.CallableStatement | Execute stored procedures |
| java.sql.CLOB | Provide support for CLOB(Character Large Object) SQL type. |
| java.sql.Date | Provide support for Date SQL type. |
| java.sql.Driver | create an instance of a driver with the DriverManager. |
| java.sql.DriverManager | This class manages database drivers. |
| java.sql.PreparedStatement | Used to create and execute parameterized query. |
| java.sql.ResultSet | It is an interface that provide methods to access the result row-by-row. |
| java.sql.Savepoint | Specify savepoint in transaction. |
| java.sql.SQLException | Encapsulate all JDBC related exception. |
| java.sql.Statement | This interface is used to execute SQL statements. |

**javax.sql package**

This package is also known as JDBC extension API. It provides classes and interface to access server-side data.

**Important classes and interface of javax.sql package**

|  |  |
| --- | --- |
| **classes/interface** | **Description** |
| javax.sql.ConnectionEvent | Provide information about occurence of event. |
| javax.sql.ConnectionEventListener | Used to register event generated by **PooledConnection**object. |
| javax.sql.DataSource | Represent the **DataSource** interface used in an application. |
| javax.sql.PooledConnection | provide object to manage connection pools. |