**Java JDBC**

JDBC stands for Java Database Connectivity. JDBC is a Java API to connect and execute the query with the database. It is a part of JavaSE (Java Standard Edition). JDBC API uses JDBC drivers to connect with the database. There are four types of JDBC drivers:

* JDBC-ODBC Bridge Driver,
* Native Driver,
* Network Protocol Driver, and
* Thin Driver

We can use JDBC API to access tabular data stored in any relational database.

By the help of JDBC API, we can save, update, delete and fetch data from the database.



The **java.sql** package contains classes and interfaces for JDBC API. A list of popular ***interfaces***of JDBC API are given below:

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

A list of popular ***classes***of JDBC API are given below:

* DriverManager class
* Blob class
* Clob class
* Types class

Why Should We Use JDBC

Before JDBC, ODBC API was the database API to connect and execute the 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).

We can use JDBC API to handle database using Java program and can perform the following activities:

1. Connect to the database.
2. Execute queries and update statements to the database.
3. Retrieve the result received from the database.

# JDBC Driver

### 1) JDBC-ODBC bridge driver

|  |
| --- |
| The JDBC-ODBC bridge driver uses ODBC driver to connect to the database.  The JDBC-ODBC bridge driver converts JDBC method calls into the ODBC function  calls. This is now discouraged because of thin driver. |



#### **In Java 8, the JDBC-ODBC Bridge has been removed.**

Oracle does not support the JDBC-ODBC Bridge from Java 8. Oracle recommends that you use JDBC drivers provided by the vendor of your database instead of the JDBC-ODBC Bridge.

### Advantages:

* easy to use.
* can be easily connected to any database.

### Disadvantages:

* Performance degraded because JDBC method call is converted into the ODBC function calls.
* The ODBC driver needs to be installed on the client machine.

### 2) Native-API driver

|  |
| --- |
| The Native API driver uses the client-side libraries of the database. The driver converts  JDBC method calls into native calls of the database API. It is not written entirely in java. |



### Advantage:

* performance upgraded than JDBC-ODBC bridge driver.

### Disadvantage:

* The Native driver needs to be installed on the each client machine.
* The Vendor client library needs to be installed on client machine.

### 3) Network Protocol driver

The Network Protocol driver uses middleware (application server) that converts JDBC calls directly or indirectly into the vendor-specific database protocol. It is fully written in java.



### Advantage:

* No client side library is required because of application server that can perform many tasks like auditing, load balancing, logging etc.

### Disadvantages:

* 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.

### 4) Thin driver

|  |
| --- |
| 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. |



### Advantage:

* Better performance than all other drivers.
* No software is required at client side or server side.

### Disadvantage:

* Drivers depend on the Database.

Java Database Connectivity with 5 Steps

|  |
| --- |
| There are 5 steps to connect any java application with the database using JDBC.  These steps are as follows:   * Register the Driver class * Create connection * Create statement * Execute queries * Close connection |



Java Database Connectivity with MySQL

To connect Java application with the MySQL database, we need to follow 5 following steps.

In this example we are using MySql as the database. So we need to know following informations for the MySQL database:

1. **Driver class:**The driver class for the mysql database is com.mysql.jdbc.Driver.
2. **Connection URL:**The connection URL for the mysql database is **jdbc:mysql://localhost:3306/database name** where jdbc is the API, mysql is the database, localhost is the server name on which mysql is running, we may also use IP address, 3306 is the port number and sonoo is the database name. We may use any database, in such case, we need to replace the sonoo with our database name.
3. **Username:**The default username for the mysql database is **root**.
4. **Password:**It is the password given by the user at the time of installing the mysql database. In this example, we are going to use root as the password.

Let's first create a table in the mysql database, but before creating table, we need to create database first.

1. create database Your Wish;
2. use Your Wish;
3. create table emp(id **int**(10),name varchar(40),age **int**(3));
4. insert into emp(id,name,age) values (101,’fgfgfg’,24);

Example to Connect Java Application with mysql database

In this example, MyWish is the database name, root is the username and password both.

**import** java.sql.\*;  “

**class** MysqlCon{

**public** **static** **void** main(String args[]){

**try**{

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

Connection con=DriverManager.getConnection(

"jdbc:mysql://localhost:3306/MyWish","root","root");

//here sonoo is database name, root is username and password

Statement stmt=con.createStatement();

ResultSet rs=stmt.executeQuery("select \* from emp");

**while**(rs.next())

System.out.println(rs.getInt(1)+"  "+rs.getString(2)+"  "+rs.getString(3));

con.close();

}**catch**(Exception e){ System.out.println(“went wrong”);}

}

}

[**JAVA MySQL Create Read Update Delete CRUD Project**](https://chillyfacts.com/java-mysql-create-read-update-delete-crud-project/)

First create a new database ‘crud’ in Mysql. Then create a table ‘user’

CREATE TABLE `user` (

`sl\_no` INT(5) NULL DEFAULT NULL,

`name` VARCHAR(100) NULL DEFAULT NULL,

`email` VARCHAR(100) NULL DEFAULT NULL

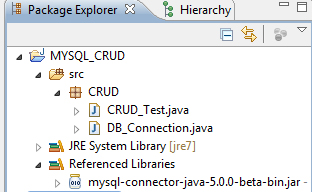
)

COLLATE='latin1\_swedish\_ci'

ENGINE=InnoDB

;

Project Structure in Eclipse IDE



Connection Class to get connection with MySql  
DB\_Connection.java

package CRUD;

import java.sql.Connection;

import java.sql.DriverManager;

public class DB\_Connection {

public static void main(String[] args) {

DB\_Connection obj\_DB\_Connection=new DB\_Connection();

System.out.println(obj\_DB\_Connection.get\_connection());

}

public Connection get\_connection(){

Connection connection=null;

try {

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

connection = DriverManager.getConnection("jdbc:mysql://localhost:3306/crud","root", "root");

} catch (Exception e) {

System.out.println(e);

}

return connection;

}

}

CRUD\_Test.java

package CRUD;

import java.sql.Connection;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

public class CRUD\_Test {

public static void main(String[] args) {

CRUD\_Test objTest=new CRUD\_Test();

//insert data

objTest.create\_data("1", "jinu", "jinu@gmail.com");

objTest.create\_data("2", "jawad", "jawad@gmail.com");

//read data

objTest.read\_data("2");

//update data

objTest.update\_data("2", "22", "jawad2", "jawad2@gmail");

//delete data

objTest.delete\_data("1");

}

public void create\_data(String sl\_no,String name,String email){

DB\_Connection obj\_DB\_Connection=new DB\_Connection();

Connection connection=obj\_DB\_Connection.get\_connection();

PreparedStatement ps=null;

try {

String query="insert into user(sl\_no,name,email) values (?,?,?)";

ps=connection.prepareStatement(query);

ps.setString(1, sl\_no);

ps.setString(2, name);

ps.setString(3, email);

System.out.println(ps);

ps.executeUpdate();

} catch (Exception e) {

System.out.println(e);

}

}

public void read\_data(String sl\_no){

DB\_Connection obj\_DB\_Connection=new DB\_Connection();

Connection connection=obj\_DB\_Connection.get\_connection();

PreparedStatement ps=null;

ResultSet rs=null;

try {

String query="select \* from user";

ps=connection.prepareStatement(query);

//ps.setString(1, sl\_no);

System.out.println(ps);

rs=ps.executeQuery();

while(rs.next()){

System.out.println("Sl no -"+rs.getString("sl\_no"));

System.out.println("name -"+rs.getString("name"));

System.out.println("email -"+rs.getString("email"));

System.out.println("---------------");

}

} catch (Exception e) {

System.out.println(e);

}

}

public void update\_data(String sl\_no,String new\_sl\_no,String name,String email){

DB\_Connection obj\_DB\_Connection=new DB\_Connection();

Connection connection=obj\_DB\_Connection.get\_connection();

PreparedStatement ps=null;

try {

String query="update user set sl\_no=?,name=?,email=? where sl\_no=?";

ps=connection.prepareStatement(query);

ps.setString(1, new\_sl\_no);

ps.setString(2, name);

ps.setString(3, email);

ps.setString(4, sl\_no);

System.out.println(ps);

ps.executeUpdate();

} catch (Exception e) {

System.out.println(e);

}

}

public void delete\_data(String sl\_no){

DB\_Connection obj\_DB\_Connection=new DB\_Connection();

Connection connection=obj\_DB\_Connection.get\_connection();

PreparedStatement ps=null;

try {

String query="delete from user where sl\_no=?";

ps=connection.prepareStatement(query);

ps.setString(1, sl\_no);

System.out.println(ps);

ps.executeUpdate();

} catch (Exception e) {

System.out.println(e);

}

}

}

