

# 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,(Out dated)
- 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. It is like Open Database Connectivity (ODBC) provided by Microsoft.

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**

JDBC Driver is a software component that enables java application to interact with the database. There are 4 types of JDBC drivers:

JDBC-ODBC bridge driver

1. Native-API driver (partially java driver)
2. Network Protocol driver (fully java driver)
3. Thin driver (fully java driver)
4. Thin driver

The thin driver converts JDBC calls directly into the vendor-specific database protocol. That is why it is known as a thin driver. It is fully written in Java language.

## **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:

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

### **1) Register the driver class**

The `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
```

#### **Example to register the `OracleDriver` class**

Here, Java program is loading oracle driver to establish database connection.

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

### **2) Create the connection object**

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

#### **Syntax of `getConnection()` method**

```
1) public static Connection getConnection(String url)throws SQLException
```

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

### **Example to establish connection with the Oracle database**

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

### **3) Create the Statement object**

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

### **Syntax of createStatement() method**

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

### **Example to create the statement object**

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

### **4) Execute the 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.

### **Syntax of executeQuery() method**

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

### **Example to execute query**

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

```
while(rs.next()){  
    System.out.println(rs.getInt(1)+" "+rs.getString(2));  
}
```

## 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
```

### Example to close connection

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

## Java Database Connectivity with Oracle

To connect java application with the oracle database, we need to follow 5 following steps. In this example, we are using Oracle 10g as the database. So we need to know following information for the oracle database:

**Driver class:** The driver class for the oracle database is **oracle.jdbc.driver.OracleDriver**.

**Connection URL:** The connection URL for the oracle10G database is **jdbc:oracle:thin:@localhost:1521:xe** where jdbc is the API, oracle is the database, thin is the driver, localhost is the server name on which oracle is running, we may also use IP address, 1521 is the port number and XE is

the Oracle service name. You may get all these information from the tnsnames.ora file.

**Username:** The default username for the oracle database is **system**.

**Password:** It is the password given by the user at the time of installing the oracle database.

## Create a Table

Before establishing connection, let's first create a table in oracle database. Following is the SQL query to create a table.

```
create table emp(id number(10),name varchar2(40),age number(3));
```

---

## Example to Connect Java Application with Oracle database

In this example, we are connecting to an Oracle database and getting data from emp table. Here, system and oracle are the username and password of the Oracle database.

```
import java.sql.*;
```

```
class OracleCon{
```

```
    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 emp");

        while(rs.next())

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

//step5 close the connection object

        con.close();

        }catch(Exception e){ System.out.println(e);}

    }

}

```

To connect java application with the Oracle database ojdbc14.jar file is required to be loaded.

**Two ways to load the jar file:**

paste the ojdbc14.jar file in jre/lib/ext folder

set classpath

### 1) paste the ojdbc14.jar file in JRE/lib/ext folder:

Firstly, search the ojdbc14.jar file then go to JRE/lib/ext folder and paste the jar file here.

### 2) set classpath:

There are two ways to set the classpath:

temporary

permanent

#### How to set the temporary classpath:

Firstly, search the ojdbc14.jar file then open command prompt and write:

```
C:>set classpath=c:\folder\ojdbc14.jar;.;
```

#### How to set the permanent classpath:

Go to environment variable then click on new tab. In variable name write `classpath` and in variable value paste the path to ojdbc14.jar by appending ojdbc14.jar;. as **C:\oracle\app\oracle\product\10.2.0\server\jdbc\lib\ojdbc14.jar;.;**

## 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:

**Driver class:** The driver class for the mysql database is **com.mysql.jdbc.Driver**.

**Connection URL:** The connection URL for the mysql database is **jdbc:mysql://localhost:3306/sonoo** 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.

**Username:** The default username for the mysql database is **root**.

**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.

```
create database sonoo;
```

```
use sonoo;
```

```
create table emp(id int(10),name varchar(40),age int(3));
```

---

### Example to Connect Java Application with mysql database

In this example, sonoo 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");

        Connection con=DriverManager.getConnection(

            "jdbc:mysql://localhost:3306/sonoo","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(e);}

}
}

```

### Two ways to load the jar file:

Paste the mysqlconnector.jar file in jre/lib/ext folder

Set classpath

#### 1) Paste the mysqlconnector.jar file in JRE/lib/ext folder:

Download the mysqlconnector.jar file. Go to jre/lib/ext folder and paste the jar file here.

## 2) Set classpath:

There are two ways to set the classpath:

temporary

permanent

### How to set the temporary classpath

open command prompt and write:

**C:>set classpath=c:\folder\mysql-connector-java-5.0.8-bin.jar;.;**

### How to set the permanent classpath

Go to environment variable then click on new tab. In variable name write `classpath` and in variable value paste the path to the mysqlconnector.jar file by appending mysqlconnector.jar;. as **C:\folder\mysql-connector-java-5.0.8-bin.jar;.;**

## Connectivity with Access without DSN

There are two ways to connect java application with the access database.

Without DSN (Data Source Name)

With DSN

Java is mostly used with Oracle, mysql, or DB2 database. So you can learn this topic only for knowledge.

---

### Example to Connect Java Application with access without DSN

In this example, we are going to connect the java program with the access database. In such case, we have created the login table in the access database. There is only one column in the table named name. Let's get all the name of the login table.

```
import java.sql.*;

class Test{

public static void main(String ar[]){

    try{        String database="student.mdb";

        //Here database exists in the current directory

        String url="jdbc:odbc:Driver={Microsoft Access Driver (*.mdb)};

        DBQ="+ database + ";DriverID=22;READONLY=true";

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

        Connection c=DriverManager.getConnection(url);

        Statement st=c.createStatement();

        ResultSet rs=st.executeQuery("select * from login");

        while(rs.next()){

            System.out.println(rs.getString(1));

        }

    }catch(Exception ee){System.out.println(ee);}

}}
```

### Example to Connect Java Application with access with DSN

Connectivity with type1 driver is not considered good. To connect java application with type1 driver, create DSN first, here we are assuming your dsn name is mydsn.

```
import java.sql.*;

class Test{

public static void main(String ar[]){

    try{

        String url="jdbc:odbc:mydsn";

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

        Connection c=DriverManager.getConnection(url);

        Statement st=c.createStatement();

        ResultSet rs=st.executeQuery("select * from login");

        while(rs.next()){

            System.out.println(rs.getString(1));

        }

    }catch(Exception ee){System.out.println(ee);}

}}
```

## DriverManager class

The DriverManager class is the component of JDBC API and also a member of the *java.sql* package. The DriverManager class acts as an interface between users and drivers. It keeps track of the drivers that are available and handles

establishing a connection between a database and the appropriate driver. It contains all the appropriate methods to register and deregister the database driver class and to create a connection between a Java application and the database. The DriverManager class maintains a list of Driver classes that have registered themselves by calling the method DriverManager.registerDriver(). Note that before interacting with a Database, it is a mandatory process to register the driver; otherwise, an exception is thrown.

### Methods of the DriverManager Class

Method	Description
<b>1)      public      static synchronized      void registerDriver(Driver driver):</b>	is used to register the given driver with DriverManager. No action is performed by the method when the given driver is already registered.
<b>2)      public      static synchronized      void deregisterDriver(Driver driver):</b>	is used to deregister the given driver (drop the driver from the list) with DriverManager. If the given driver has been removed from the list, then no action is performed by the method.
<b>3)      public      static Connection getConnection(String url)                      throws SQLException:</b>	is used to establish the connection with the specified url. The SQLException is thrown when the corresponding Driver class of the given database is not registered with the DriverManager.

<b>4)      public      static Connection getConnection(String url,String userName,String password)      throws SQLException:</b>	<p>is used to establish the connection with the specified url, username, and password. The SQLException is thrown when the corresponding Driver class of the given database is not registered with the DriverManager.</p>
<b>5) public static Driver getDriver(String url)</b>	<p>Those drivers that understand the mentioned URL (present in the parameter of the method) are returned by this method provided those drivers are mentioned in the list of registered drivers.</p>
<b>6)   pubic   static   int getLoginTimeout()</b>	<p>The duration of time a driver is allowed to wait in order to establish a connection with the database is returned by this method.</p>
<b>7)   pubic   static   void setLoginTimeout(int sec)</b>	<p>The method provides the time in seconds. sec mentioned in the parameter is the maximum time that a driver is allowed to wait in order to establish a connection with the database. If 0 is passed in the parameter of this method, the driver will have to wait infinitely while trying to establish the connection with the database.</p>

<b>8)      public      static Connection getConnection(String URL, Properties prop) throws SQLException</b>	A connection object is returned by this method after creating a connection to the database present at the mentioned URL, which is the first parameter of this method. The second parameter, which is "prop", fetches the authentication details of the database (username and password.). Similar to the other variation of the getConnection() method, this method also throws the SQLException, when the corresponding Driver class of the given database is not registered with the DriverManager.
---	---

## Connection interface

A Connection is a session between a Java application and a database. It helps to establish a connection with the database.

The Connection interface is a factory of Statement, PreparedStatement, and DatabaseMetaData, i.e., an object of Connection can be used to get the object of Statement and DatabaseMetaData. The Connection interface provide many methods for transaction management like commit(), rollback(), setAutoCommit(), setTransactionIsolation(), etc.

**By default, connection commits the changes after executing queries.**

### Commonly used methods of Connection interface:

1) **public Statement createStatement():** creates a statement object that can be used to execute SQL queries.



2) **public Statement createStatement(int resultSetType,int resultSetConcurrency):** Creates a Statement object that will generate ResultSet objects with the given type and concurrency.

3) **public void setAutoCommit(boolean status):** is used to set the commit status. By default, it is true.

4) **public void commit():** saves the changes made since the previous commit/rollback is permanent.

5) **public void rollback():** Drops all changes made since the previous commit/rollback.

6) **public void close():** closes the connection and Releases a JDBC resources immediately.

## Connection Interface Fields

There are some common Connection interface constant fields that are present in the Connect interface. These fields specify the isolation level of a transaction.

**TRANSACTION\_NONE:** No transaction is supported, and it is indicated by this constant.

**TRANSACTION\_READ\_COMMITTED:** It is a constant which shows that the dirty reads are not allowed. However, phantom reads and non-repeatable reads can occur.

**TRANSACTION\_READ\_UNCOMMITTED:** It is a constant which shows that dirty reads, non-repeatable reads, and phantom reads can occur.

**TRANSACTION\_REPEATABLE\_READ:** It is a constant which shows that the non-repeatable reads and dirty reads are not allowed. However, phantom reads and can occur.

**TRANSACTION\_SERIALIZABLE:** It is a constant which shows that the non-repeatable reads, dirty reads as well as the phantom reads are not allowed.

## Statement interface

The **Statement interface** provides methods to execute queries with the database. The statement interface is a factory of ResultSet i.e. it provides factory method to get the object of ResultSet.

### Commonly used methods of Statement interface:

The important methods of Statement interface are as follows:

- 1) **public ResultSet executeQuery(String sql):** is used to execute SELECT query. It returns the object of ResultSet.
- 2) **public int executeUpdate(String sql):** is used to execute specified query, it may be create, drop, insert, update, delete etc.
- 3) **public boolean execute(String sql):** is used to execute queries that may return multiple results.
- 4) **public int[] executeBatch():** is used to execute batch of commands.

### Example of Statement interface

Let's see the simple example of Statement interface to insert, update and delete the record.

```
import java.sql.*;
```

```
class FetchRecord{
```

```
    public static void main(String args[])throws Exception{
```

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

```

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

        Statement stmt=con.createStatement();

//stmt.executeUpdate("insert into emp765 values(33,'Irfan',50000)");

//int      result=stmt.executeUpdate("update      emp765      set
name='Vimal',salary=10000 where id=33");

int result=stmt.executeUpdate("delete from emp765 where id=33");

System.out.println(result+" records affected");

con.close();

}}

```

## ResultSet interface

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

**By default, ResultSet object can be moved forward only and it is not updatable.**

But we can make this object to move forward and backward direction by passing either TYPE\_SCROLL\_INSENSITIVE or TYPE\_SCROLL\_SENSITIVE in createStatement(int,int) method as well as we can make this object as updatable by:

```

Statement      stmt      =
con.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,

```

ResultSet.CONCUR\_UPDATABLE);

### Commonly used methods of ResultSet interface

1) public boolean next():	is used to move the cursor to the one row next from the current position.
2) public boolean previous():	is used to move the cursor to the one row previous from the current position.
3) public boolean first():	is used to move the cursor to the first row in result set object.
4) public boolean last():	is used to move the cursor to the last row in result set object.
5) public boolean absolute(int row):	is used to move the cursor to the specified row number in the ResultSet object.
6) public boolean relative(int row):	is used to move the cursor to the relative row number in the ResultSet object, it may be positive or negative.
7) public int getInt(int columnIndex):	is used to return the data of specified column index of the current row as int.

8) public int getInt(String columnName):	is used to return the data of specified column name of the current row as int.
9) public String getString(int columnIndex):	is used to return the data of specified column index of the current row as String.
10) public String getString(String columnName):	is used to return the data of specified column name of the current row as String.

### Example of Scrollable ResultSet

Let's see the simple example of ResultSet interface to retrieve the data of 3rd row.

```
import java.sql.*;
```

```
class FetchRecord{
```

```
public static void main(String args[])throws Exception{
```

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

```
Connection
```

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

```
Statement
```

```
stmt=con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_UPDATABLE);
```

```
ResultSet rs=stmt.executeQuery("select * from emp765");
```

```
//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 subinterface of Statement. It is used to execute parameterized query.

Let's see the example of parameterized query:

```
String sql="insert into emp values(?,?,?)";
```

As you can see, we are passing parameter (?) for the values. Its value will be set by calling the setter methods of PreparedStatement.

### Why use PreparedStatement?

**Improves performance:** The performance of the application will be faster if you use PreparedStatement interface because query is compiled only once.

---

### How to get the instance of PreparedStatement?

The prepareStatement() method of Connection interface is used to return the object of PreparedStatement. Syntax:

```
public PreparedStatement prepareStatement(String query)throws SQLException{
```

---

### Methods of PreparedStatement interface

The important methods of PreparedStatement interface are given below:

Method	Description
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.
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.

### Example of PreparedStatement interface that inserts the record

First of all create table as given below:

```
create table emp(id number(10),name varchar2(50));
```

Now insert records in this table by the code given below:

```
import java.sql.*;

class InsertPrepared{

public static void main(String args[]){

    try{

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

        Connection
        con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:152
        1:xe","system","oracle");

        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");

        con.close();

    }catch(Exception e){ System.out.println(e);}

}
```



```
}
```

---

#### Example of PreparedStatement interface that updates the record

```
PreparedStatement stmt=con.prepareStatement("update emp set name=?  
where id=?");  
  
stmt.setString(1,"Sonoo");//1 specifies the first parameter in the query i.e. name  
  
stmt.setInt(2,101);  
  
int i=stmt.executeUpdate();  
  
System.out.println(i+" records updated");
```

---

#### Example of PreparedStatement interface that deletes the record

```
PreparedStatement stmt=con.prepareStatement("delete from emp where id=?");  
  
stmt.setInt(1,101);  
  
int i=stmt.executeUpdate();  
  
System.out.println(i+" records deleted");
```

---

#### Example of PreparedStatement interface that retrieve the records of a table

```
PreparedStatement stmt=con.prepareStatement("select * from emp");  
  
ResultSet rs=stmt.executeQuery();  
  
while(rs.next()){
```

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

---

### Example of PreparedStatement to insert records until user press n

```
import java.sql.*;  
import java.io.*;  
class RS{  
    public static void main(String args[])throws Exception{  
        Class.forName("oracle.jdbc.driver.OracleDriver");  
        Connection  
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:152  
1:xe","system","oracle");  
        PreparedStatement ps=con.prepareStatement("insert into emp130  
values(?,?,?)");  
        BufferedReader br=new BufferedReader(new  
InputStreamReader(System.in));  
        do{  
            System.out.println("enter id:");  
            int id=Integer.parseInt(br.readLine());  
            System.out.println("enter name:");  
            String name=br.readLine();  
            System.out.println("enter salary:");
```

```
float salary=Float.parseFloat(br.readLine());

ps.setInt(1,id);

ps.setString(2,name);

ps.setFloat(3,salary);

int i=ps.executeUpdate();

System.out.println(i+" records affected");

System.out.println("Do you want to continue: y/n");

String s=br.readLine();

if(s.startsWith("n")){

break;

}

}while(true);

con.close();

}}
```

## Java ResultSetMetaData Interface

The metadata means data about data i.e. we can get further information from the data.

If you have to get metadata of a table like total number of column, column name, column type etc. , ResultSetMetaData interface is useful because it provides methods to get metadata from the ResultSet object.

## Commonly used methods of ResultSetMetaData interface

Method	Description
<code>public int getColumnCount()throws SQLException</code>	it returns the total number of columns in the ResultSet object.
<code>public String getColumnName(int index)throws SQLException</code>	it returns the column name of the specified column index.
<code>public String getColumnName(int index)throws SQLException</code>	it returns the column type name for the specified index.
<code>public String getTableName(int index)throws SQLException</code>	it returns the table name for the specified column index.

### How to get the object of ResultSetMetaData:

The `getMetaData()` method of `ResultSet` interface returns the object of `ResultSetMetaData`. Syntax:

```
public ResultSetMetaData getMetaData()throws SQLException
```

---

### Example of ResultSetMetaData interface :

```
import java.sql.*;
```

```

class Rsmd{

    public static void main(String args[]){

        try{

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

            Connection con=DriverManager.getConnection(

                "jdbc:oracle:thin:@localhost:1521:xe","system","oracle");

            PreparedStatement ps=con.prepareStatement("select * from emp");

            ResultSet rs=ps.executeQuery();

            ResultSetMetaData rsmd=rs.getMetaData();

            System.out.println("Total columns: "+rsmd.getColumnCount());

            System.out.println("Column      Name      of      1st      column:
"+rsmd.getColumnName(1));

            System.out.println("Column      Type      Name      of      1st      column:
"+rsmd.getColumnTypeName(1));

            con.close();

        }catch(Exception e){ System.out.println(e);}

    }

}

```

Column Name of 1st column: ID

