Java JDBC Tutorial

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 have discussed the above four drivers in the next chapter.

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 current version of JDBC is 4.3. It is the stable release since 21st September, 2017. It is based on the X/Open SQL Call Level Interface. 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.

Do You Know

* How to connect Java application with Oracle and Mysql database using JDBC?
* What is the difference between Statement and PreparedStatement interface?
* How to print total numbers of tables and views of a database using JDBC?
* How to store and retrieve images from Oracle database using JDBC?
* How to store and retrieve files from Oracle database using JDBC?

What is API

API (Application programming interface) is a document that contains a description of all the features of a product or software. It represents classes and interfaces that software programs can follow to communicate with each other. An API can be created for applications, libraries, operating systems, etc.

# JDBC Driver

1. [JDBC Drivers](https://www.javatpoint.com/jdbc-driver)
   1. [JDBC-ODBC bridge driver](https://www.javatpoint.com/jdbc-driver#driver1)
   2. [Native-API driver](https://www.javatpoint.com/jdbc-driver#driver2)
   3. [Network Protocol driver](https://www.javatpoint.com/jdbc-driver#driver3)
   4. [Thin driver](https://www.javatpoint.com/jdbc-driver#driver4)

|  |
| --- |
| JDBC Driver is a software component that enables java application to interact with the database. There are 4 types of JDBC drivers:   1. JDBC-ODBC bridge driver 2. Native-API driver (partially java driver) 3. Network Protocol driver (fully java driver) 4. Thin driver (fully java 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

1. [5 Steps to connect to the database in java](https://www.javatpoint.com/steps-to-connect-to-the-database-in-java)
   1. [Register the driver class](https://www.javatpoint.com/steps-to-connect-to-the-database-in-java#step1)
   2. [Create the connection object](https://www.javatpoint.com/steps-to-connect-to-the-database-in-java#step2)
   3. [Create the Statement object](https://www.javatpoint.com/steps-to-connect-to-the-database-in-java#step3)
   4. [Execute the query](https://www.javatpoint.com/steps-to-connect-to-the-database-in-java#step4)
   5. [Close the connection object](https://www.javatpoint.com/steps-to-connect-to-the-database-in-java#step5)

|  |
| --- |
| 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 |

### 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

1. **public** **static** **void** forName(String className)**throws** ClassNotFoundException

#### Note: Since JDBC 4.0, explicitly registering the driver is optional. We just need to put vender's Jar in the classpath, and then JDBC driver manager can detect and load the driver automatically.

### Example to register the OracleDriver class

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

1. 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. 1) **public** **static** Connection getConnection(String url)**throws** SQLException
2. 2) **public** **static** Connection getConnection(String url,String name,String password)
3. **throws** SQLException

### Example to establish connection with the Oracle database

1. Connection con=DriverManager.getConnection(
2. "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 statement is responsible to execute queries with the database. |

### Syntax of createStatement() method

1. **public** Statement createStatement()**throws** SQLException

### Example to create the statement object

1. 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

1. **public** ResultSet executeQuery(String sql)**throws** SQLException

### Example to execute query

1. ResultSet rs=stmt.executeQuery("select \* from emp");
3. **while**(rs.next()){
4. System.out.println(rs.getInt(1)+" "+rs.getString(2));
5. }

### 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

1. **public** **void** close()**throws** SQLException

### Example to close connection

1. con.close();

#### Note: Since Java 7, JDBC has ability to use try-with-resources statement to automatically close resources of type Connection, ResultSet, and Statement.

It avoids explicit connection closing step.

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:   1. **Driver class:**The driver class for the oracle database is **oracle.jdbc.driver.OracleDriver**. 2. **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. 3. **Username:**The default username for the oracle database is **system**. 4. **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. |

1. 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.

1. **import** java.sql.\*;
2. **class** OracleCon{
3. **public** **static** **void** main(String args[]){
4. **try**{
5. //step1 load the driver class
6. Class.forName("oracle.jdbc.driver.OracleDriver");
8. //step2 create  the connection object
9. Connection con=DriverManager.getConnection(
10. "jdbc:oracle:thin:@localhost:1521:xe","system","oracle");
12. //step3 create the statement object
13. Statement stmt=con.createStatement();
15. //step4 execute query
16. ResultSet rs=stmt.executeQuery("select \* from emp");
17. **while**(rs.next())
18. System.out.println(rs.getInt(1)+"  "+rs.getString(2)+"  "+rs.getString(3));
20. //step5 close the connection object
21. con.close();
23. }**catch**(Exception e){ System.out.println(e);}
25. }
26. }

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [**next →**](https://www.javatpoint.com/Connection-interface)[**← prev**](https://www.javatpoint.com/example-to-connect-to-the-mysql-database)  DriverManager class  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. The DriverManager class maintains a list of Driver classes that have registered themselves by calling the method DriverManager.registerDriver().  Useful methods of DriverManager class   |  |  | | --- | --- | | **Method** | **Description** | | 1) public static void registerDriver(Driver driver): | is used to register the given driver with DriverManager. | | 2) public static void deregisterDriver(Driver driver): | is used to deregister the given driver (drop the driver from the list) with DriverManager. | | 3) public static Connection getConnection(String url): | is used to establish the connection with the specified url. | | 4) public static Connection getConnection(String url,String userName,String password): | is used to establish the connection with the specified url, username and password. | |

# Connection interface

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. The Connection interface provide many methods for transaction management like commit(), rollback() 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 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. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| [**next →**](https://www.javatpoint.com/ResultSet-interface)[**← prev**](https://www.javatpoint.com/Connection-interface)  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.   1. **import** java.sql.\*; 2. **class** FetchRecord{ 3. **public** **static** **void** main(String args[])**throws** Exception{ 4. Class.forName("oracle.jdbc.driver.OracleDriver"); 5. Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","oracle"); 6. Statement stmt=con.createStatement(); 8. //stmt.executeUpdate("insert into emp765 values(33,'Irfan',50000)"); 9. //int result=stmt.executeUpdate("update emp765 set name='Vimal',salary=10000 where id=33"); 10. **int** result=stmt.executeUpdate("delete from emp765 where id=33"); 11. System.out.println(result+" records affected"); 12. con.close(); 13. }} |

# 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:

1. Statement stmt = con.createStatement(ResultSet.TYPE\_SCROLL\_INSENSITIVE,
2. 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.

1. **import** java.sql.\*;
2. **class** FetchRecord{
3. **public** **static** **void** main(String args[])**throws** Exception{
5. Class.forName("oracle.jdbc.driver.OracleDriver");
6. Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","oracle");
7. Statement stmt=con.createStatement(ResultSet.TYPE\_SCROLL\_SENSITIVE,ResultSet.CONCUR\_UPDATABLE);
8. ResultSet rs=stmt.executeQuery("select \* from emp765");
10. //getting the record of 3rd row
11. rs.absolute(3);
12. System.out.println(rs.getString(1)+" "+rs.getString(2)+" "+rs.getString(3));
14. con.close();
15. }}