Java JDBC

Java JDBC is a java API to connect and execute query with the database. JDBC API uses jdbc drivers to connect with the database.

JDBC (Java Database Connectivity) 

Why use JDBC

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

What is API

API (Application programming interface) is a document that contains 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-ODBC bridge driver](https://www.javatpoint.com/jdbc-driver#driver1)(Traditional Driver no use of today(obselete))
  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. |



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 mtasks 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 depends on the Database.

5 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)

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

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 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();

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().

Commonly used methods of DriverManager class:

|  |  |
| --- | --- |
| 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. |

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

**import** java.sql.\*;

**class** FetchRecord{

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

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

Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:e","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=10000where 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:1521: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 i=?");

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:1521: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** |
| public int getColumnCount()throws SQLException | it returns the total number of columns in the ResultSet object. |
| public String getColumnName(int index)throws SQLException | it returns the column name of the specified column index. |
| public String getColumnTypeName(int index)throws SQLException | it returns the column type name for the specified index. |
| public String getTableName(int index)throws SQLException | 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);}

}

}

Java DatabaseMetaData interface

DatabaseMetaData interface provides methods to get meta data of a database such as database product name, database product version, driver name, name of total number of tables, name of total number of views etc.

Commonly used methods of DatabaseMetaData interface

* **public String getDriverName()throws SQLException:**it returns the name of the JDBC driver.
* **public String getDriverVersion()throws SQLException:**it returns the version number of the JDBC driver.
* **public String getUserName()throws SQLException:**it returns the username of the database.
* **public String getDatabaseProductName()throws SQLException:**it returns the product name of the database.
* **public String getDatabaseProductVersion()throws SQLException:**it returns the product version of the database.
* **public ResultSet getTables(String catalog, String schemaPattern, String tableNamePattern, String[] types)throws SQLException:**it returns the description of the tables of the specified catalog. The table type can be TABLE, VIEW, ALIAS, SYSTEM TABLE, SYNONYM etc.

How to get the object of DatabaseMetaData:

The getMetaData() method of Connection interface returns the object of DatabaseMetaData. Syntax:

**public** DatabaseMetaData getMetaData()**throws** SQLException

Simple Example of DatabaseMetaData interface :

**import** java.sql.\*;

**class** Dbmd{

**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");

DatabaseMetaData dbmd=con.getMetaData();

System.out.println("Driver Name: "+dbmd.getDriverName());

System.out.println("Driver Version: "+dbmd.getDriverVersion());

System.out.println("UserName: "+dbmd.getUserName());

System.out.println("Database Product Name: "+dbmd.getDatabaseProductName());

System.out.println("Database Product Version: "+dbmd.getDatabaseProductVersion());

con.close();

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

}

}