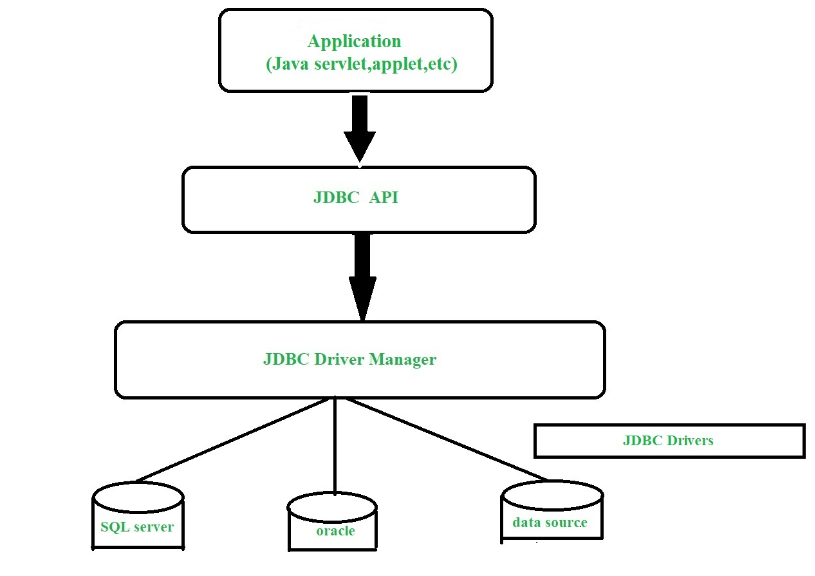
Everything about JDBC

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. 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

An ODBC driver uses the Open Database Connectivity (ODBC) interface by Microsoft that allows applications to access data in database management systems (DBMS) using SQL as a standard for accessing the data.

### **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.

### **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.

# 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

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

*Statement stmt=con.createStatement();*

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

*con.close();*

# 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.

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.

# 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:

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.

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();

7.

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

ResultSet interface **represents the result set of a database query**. A ResultSet object maintains a cursor that points to the current row in the result set. The term "result set" refers to the row and column data contained in a ResultSet object. Navigational methods − Used to move the cursor around.

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. |

# PreparedStatement interface

The PreparedStatement interface is a subinterface of Statement. It is used to execute 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.

#### **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{}

PreparedStatement stmt=con.prepareStatement("insert into Emp values(?,?)");

stmt.setInt(1,101);//1 specifies the first parameter in the query

stmt.setString(2,"Ratan");

Example of PreparedStatement interface that updates the record

1.  PreparedStatement stmt=con.prepareStatement("update emp set name=? where id=?");

2.  stmt.setString(1,"Sonoo");//1 specifies the first parameter in the query i.e. name

3.  stmt.setInt(2,101);

|  |  |
| --- | --- |
| **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. |

# Difference between Statement and PreparedStatement

**1.Statement:**   
It is used for accessing your database. Statement interface cannot accept parameters and useful when you are using static SQL statements at runtime. If you want to run [SQL](https://www.geeksforgeeks.org/sql-tutorial/) query only once then this interface is preferred over Prepared Statement.

It is base interface.

**2.PreparedStatement:**   
It is used when you want to use SQL statements many times. The Prepared Statement interface accepts input parameters at runtime.

|  |
| --- |
| It extends statement interface. |
|  |