Topics to cover:

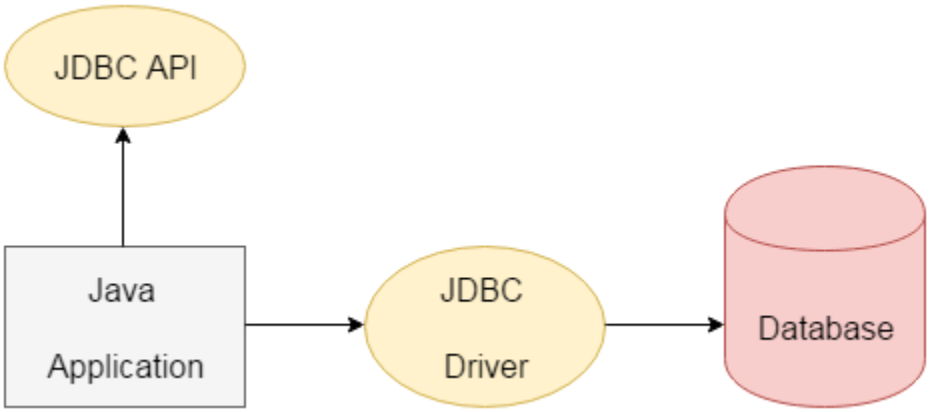
1. Java Database connection

1. Java Database connection

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:

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:

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).

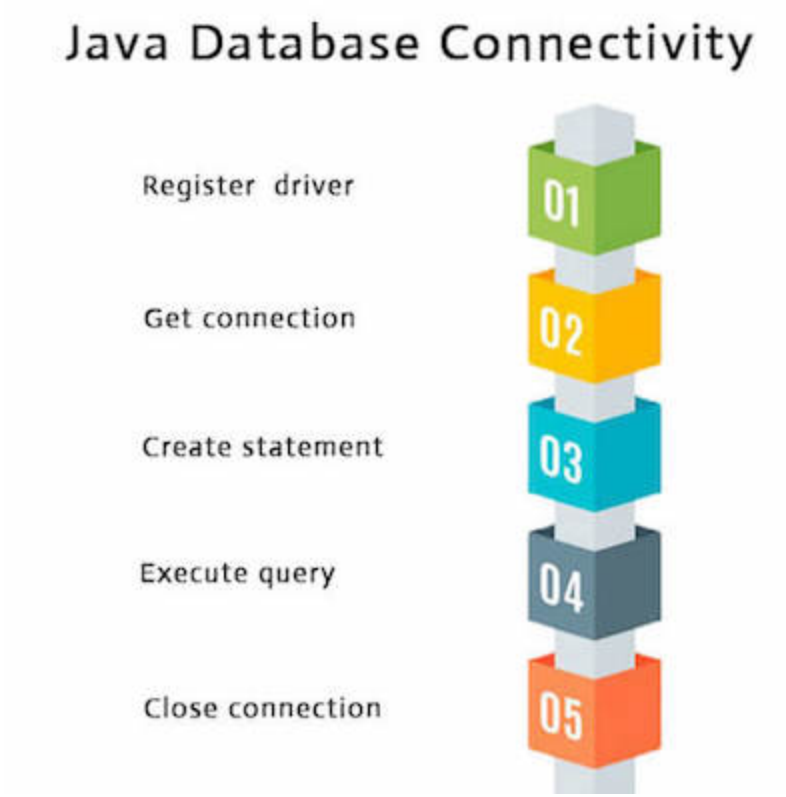
We can use JDBC API to handle database using Java program and can perform the following activities:

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

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



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
```

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 establish database connection.

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

---

2) Create the connection object

The **getConnection()** method of the 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)
3. **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 statements 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();
```

> 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 MySQL

To connect a 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/employee** 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 employee is the database name. We may use any database, in such case, we need to replace the employee 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 a table, we need to create a database first.

```
create database employee;
```

```
use employee;

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

Example to Connect Java Application with mysql database

In this example, employee 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/employee","root","root");

//here employee 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);}
```

```
	}

	}
```

The driver in the above code may be obsolete. So you can also write the following updated code. Please make sure that the driver is compatible with your database.

```
import java.sql.*;
public class MysqlCon2 {
	public static void main(String arg[])
	{
		Connection connection = null;
		try {
			// below two lines are used for connectivity.
			Class.forName("com.mysql.cj.jdbc.Driver");
			connection = DriverManager.getConnection(
				"jdbc:mysql://localhost:3306/employee",
				"root", "1234");
			// employee is database
			// root is name of database
			// mydb1234 user is password of database
			Statement statement;
			statement = connection.createStatement();
			ResultSet resultSet;
			resultSet = statement.executeQuery(
				"select * from designation");
			int code;
			String title;
			while (resultSet.next()) { //move cursor to the next row
				code = resultSet.getInt("code");
				title = resultSet.getString("title").trim();
				System.out.println("Code : " + code
							+ " Title : " + title);
			}
			resultSet.close();
			statement.close();
```

```
                connection.close();
            }
            catch (Exception exception) {
                System.out.println(exception);
            }
        } // function ends
} // class ends
```

If you cannot remember the column/attribute names, you can simply use the integer values 1, 2 … to get the values. For example, you can use **resultSet.getInt(1)** to retrieve the element of the first column of a certain row.