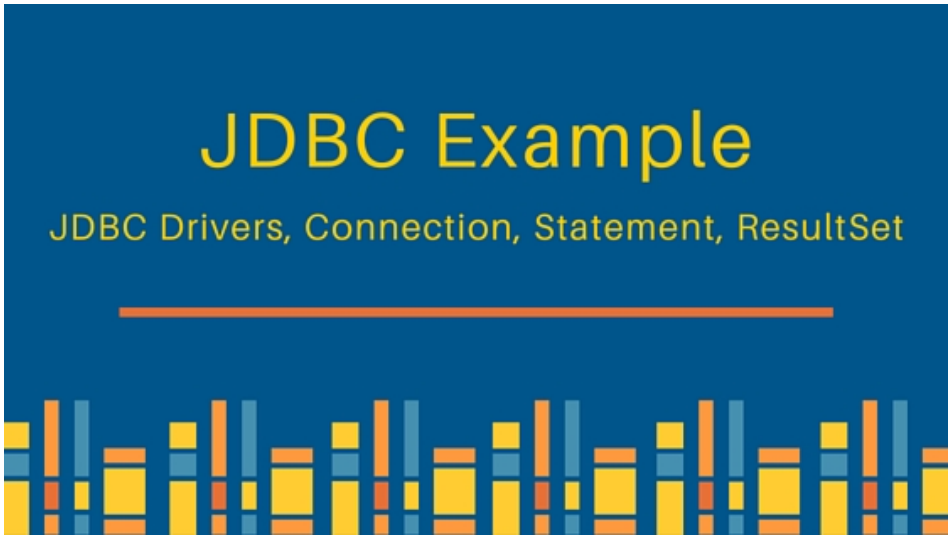
JDBC Example – MySQL, Ora

APRIL 2, 2018 BY PANKAJ — 16 COMMENTS

Welcome to the JDBC Example. Java Database Connectivity or JDBC API provides industry-standard and database-independent connectivity between the java applications and relational database servers. Just like java programs that we can "write once and run everywhere", JDBC provides framework to connect to relational databases from java programs.

Table of Contents [hide]

JDBC Example



JDBC API is used to achieve following tasks:

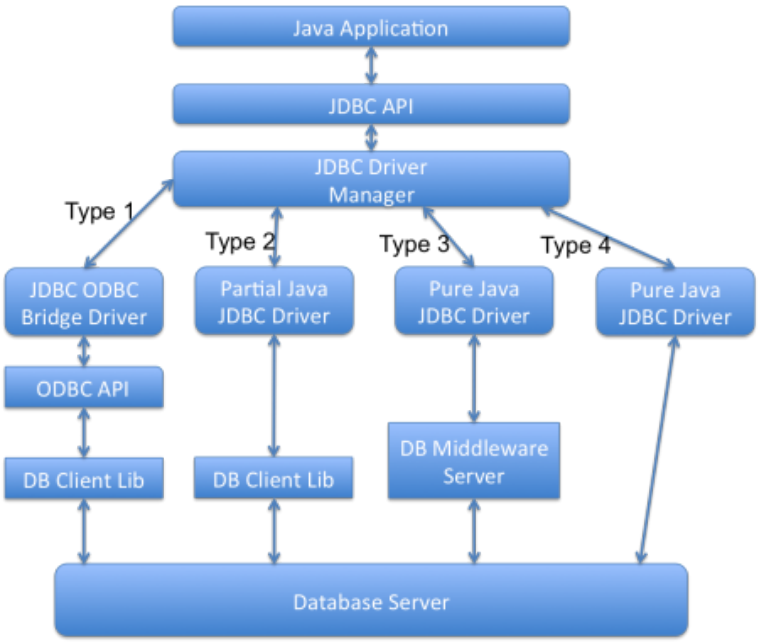
- Establishing a connection to relational Database servers like Oracle, MySQL etc. JDBC API doesn't provide framework to connect to NoSQL databases like MongoDB.
- Send SQL queries to the Connection to be executed at database server.
- Process the results returned by the execution of the query.

We will look into JDBC MySQL Example as well as JDBC Oracle Example. We will read database configuration from property file to make our code loosely coupled from database drivers.

JDBC Driver

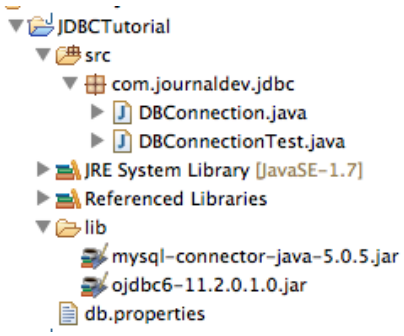
JDBC API consists of two parts – first part is the JDBC API to be used by the application programmers. Second part is the low-level API to connect to database server. First part of JDBC API is part of standard java packages in `java.sql` package.

For second part there are four different types of JDBC drivers:



1. **JDBC-ODBC Bridge plus ODBC Driver** (Type 1): This driver uses ODBC driver to connect to database servers. We should have ODBC drivers installed in the machines from where we want to connect to database, that's why this driver is almost obsolete and should be used only when other options are not available.
2. **Native API partly Java technology-enabled driver** (Type 2): This type of driver converts JDBC class to the client API for the RDBMS servers. We should have database client API installed at the machine from which we want to make database connection. Because of extra dependency on database client API drivers, this is also not preferred driver.
3. **Pure Java Driver for Database Middleware** (Type 3): This type of driver sends the JDBC calls to a middleware server that can connect to different type of databases. We should have a middleware server installed to work with this kind of driver. This adds to extra network calls and slow performance. Hence this is also not widely used JDBC driver.
4. **Direct-to-Database Pure Java Driver** (Type 4): This is the preferred driver because it converts the JDBC calls to the network protocol understood by the database server. This solution doesn't require any extra APIs at the client side and suitable for database connectivity over the network. However for this solution, we should use database specific drivers, for example OJDBC jars provided by Oracle for Oracle DB and MySQL Connector/J for MySQL databases.

Let's create a simple JDBC Example Project and see how JDBC API helps us in writing loosely-coupled code for database connectivity.



Before starting with the jdbc example, we need to do some prep work to have some data in the database servers to query.

Installing the database servers is not in the scope of this tutorial, so I will assume that you have database servers installed.

We will write program to connect to database server and run a simple jdbc query and process the results. For showing how we can achieve loose-coupling in connecting to databases using JDBC API, I will use Oracle and MySQL database systems.

Run below SQL scripts to create the table and insert some dummy values in the table.

```
--mysql create table
create table Users(
  id  int(3) primary key,
  name varchar(20),
  email varchar(20),
  country varchar(20),
  password varchar(20)
  );

--oracle create table
create table Users(
  id  number(3) primary key,
  name varchar2(20),
  email varchar2(20),
  country varchar2(20),
  password varchar2(20)
  );

--insert rows
INSERT INTO Users (id, name, email, country, password)
VALUES (1, 'Pankaj', 'pankaj@apple.com', 'India', 'pankaj123');
INSERT INTO Users (id, name, email, country, password)
```

Notice that datatypes in Oracle and MySQL databases are different, that's why I have provided two different SQL DDL queries to create Users table. However both the databases confirms to SQL language, so insert queries are same for both the database tables.

JDBC Example – Database Drivers

As you can see in the project image, I have both MySQL (mysql-connector-java-5.0.5.jar) and Oracle (ojdbc6-11.2.0.1.0.jar) type-4 drivers in the lib directory and added to the project build path. Make sure you are using the correct version of the java drivers according to your database server installation version. Usually these

jars shipped with the installer, so you can find them in the installation package. If you have maven based application, you can use below dependencies too.

```
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>5.0.5</version>
</dependency>

<!-- You need to install ojdbc6 jar manually to your maven repository -->
<dependency>
    <groupId>com.oracle</groupId>
    <artifactId>ojdbc6</artifactId>
    <version>11.2.0.1.0</version>
</dependency>
```

JDBC Database Configuration Property File

We will read the database configuration details from the property files. This will help us in switching from Oracle to MySQL database easily and vice versa. All we would need is to change the property details.

```
#mysql DB properties
#DB_DRIVER_CLASS=com.mysql.jdbc.Driver
#DB_URL=jdbc:mysql://localhost:3306/UserDB
#DB_USERNAME=pankaj
#DB_PASSWORD=pankaj123

#Oracle DB Properties
DB_DRIVER_CLASS=oracle.jdbc.driver.OracleDriver
DB_URL=jdbc:oracle:thin:@localhost:1571:MyDBSID
DB_USERNAME=scott
DB_PASSWORD=tiger
```

Database configurations are the most important details when using JDBC API. The first thing we should know is the Driver class to use. For Oracle database, driver class is `oracle.jdbc.driver.OracleDriver`. For MySQL database, driver class is `com.mysql.jdbc.Driver`. You will find these driver classes in their respective driver jar files. Both of these implement JDBC `java.sql.Driver` interface.

The second important part is the database connection URL string. Every database driver has it's own way to configure the database URL but all of them have host, port and Schema details in the connection URL.

MySQL database connection String format is `jdbc:mysql://<HOST>:<PORT>/<SCHEMA>`.

Oracle database connection string format is `jdbc:oracle:thin:@<HOST>:<PORT>:<SID>`.

The other important details are database username and password details to be used for connecting to the database server.

JDBC Example Program

Let's see a simple jdbc example program to see how we can read above properties and create database connection.

```
        public static Connection getConnection() {
                Properties props = new Properties();
                FileInputStream fis = null;
                Connection con = null;
                try {
                        fis = new FileInputStream("db.properties");
                        props.load(fis);

                        // load the Driver Class
                        Class.forName(props.getProperty("DB_DRIVER_CLASS"));

                        // create the connection now
                        con =
DriverManager.getConnection(props.getProperty("DB_URL"),
                                        props.getProperty("DB_USERNAME"),
                                        props.getProperty("DB_PASSWORD"));
                } catch (IOException | ClassNotFoundException | SQLException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }
                return con;
        }
}
```

Above jdbc example program is really simple. First we are reading database configuration details from the property file and then loading the JDBC driver and using DriverManager to create the connection. Notice that this code use only Java JDBC API classes and there is no way to know that it's connecting to which type of database. This is also a great example of **writing code for interfaces** methodology.

The important code to notice is the `Class.forName()` method call, this is the Java Reflection method to create the instance of the given class. You might wonder why we are using Reflection and not `new` operator to create the object and why we are just creating the object and not using it.

The first reason is that using reflection to create instance helps us in writing loosely-coupled code that we can't achieve if we are using new operator. In that case, we could not switch to different database without making corresponding code changes.

The reason for not using the object is because we are not interested in creating the object. The main motive is to load the class into memory, so that the driver class can register itself to the DriverManager. If you will look into the Driver classes implementation, you will find that they have static block where they are registering themselves to DriverManager.

`oracle.jdbc.driver.OracleDriver.java` snippet:

```
static
  {
    try
    {
      if (defaultDriver == null)
      {
        defaultDriver = new oracle.jdbc.OracleDriver();
        DriverManager.registerDriver(defaultDriver);
      }
        //some code omitted for clarity
        }
}
```

`com.mysql.jdbc.Driver.java` snippet:

Copy

```
static
  {
    try
    {
      DriverManager.registerDriver(new Driver());
    } catch (SQLException E) {
      throw new RuntimeException("Can't register driver!");
    }
  }
```

This is a great example where we are making our code loosely-coupled with the use of reflection API. So basically we are doing following things using Class.forName() method call.

```
Driver driver = new OracleDriver();
DriverManager.registerDriver(driver);
```

DriverManager.getConnection() method uses the registered JDBC drivers to create the database connection. This method throws `java.sql.SQLException` if there is any problem in getting the database connection.

Now let's write a simple jdbc example test program to use the database connection and run simple query.

JDBC Statement and ResultSet

Here is a simple jdbc example program where we are using the JDBC Connection to execute SQL query against the database and then processing the result set.

```
package com.journaldev.jdbc;

import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class DBConnectionTest {

	private static final String QUERY = "select id,name,email,country,password
from Users";

	public static void main(String[] args) {

		//using try-with-resources to avoid closing resources (boiler plate
code)
		try(Connection con = DBConnection.getConnection();
				Statement stmt = con.createStatement();
				ResultSet rs = stmt.executeQuery(QUERY)) {

			while(rs.next()){
				int id = rs.getInt("id");
```

Notice that we are using Java 7 try-with-resources feature to make sure that resources are closed as soon as we are out of try-catch block.

JDBC Connection, Statement and ResultSet are expensive resources and we should close them as soon as we are finished using them.

`Connection.createStatement()` is used to create the Statement object and then `executeQuery()` method is used to run the query and get the result set object.

First call to ResultSet `next()` method call moves the cursor to the first row and subsequent calls moves the cursor to next rows in the result set. If there are no more rows then it returns false and come out of the while loop. We are using result set `getXXX()` method to get the columns value and then writing them to the console.

When we run above jdbc example test program, we get following output.

```
1,Pankaj,pankaj@apple.com,India,pankaj123
4,David,david@gmail.com,USA,david123
5,Raman,raman@google.com,UK,raman123
```

Just uncomment the MySQL database configuration properties from db.properties file and comment the Oracle database configuration details to switch to MySQL database. Since the data is same in both Oracle and MySQL database Users table, you will get the same output.

That's all for the JDBC example. You can see that JDBC API helps us in writing database driver independent code and makes it easier to switch to other relational databases.

Download the project from below link and run different scenarios for better understanding.

Download JDBC Example Project

« PREVIOUS
Serialization in Java – Java Serialization

NEXT »
Java JDBC Transaction Management and Savepoint

About Pankaj

If you have come this far, it means that you liked what you are reading. Why not reach little more and connect with me directly on **Google Plus**, **Facebook** or **Twitter**. I would love to hear your thoughts and opinions on my articles directly.

Recently I started creating video tutorials too, so do check out my videos on **Youtube**.

FILED UNDER: DATABASE, JAVA

Comments

HamzeH says
JANUARY 26, 2017 AT 12:34 AM
thanks alot

Reply

Lakhveer Singh says
FEBRUARY 22, 2016 AT 3:00 AM
Ther was way to add username password.
First i have connected to localhost database using host as 127.0.0.1. now i am trying to connect to 54.35.xx.xxx Mysql database but i m getting this error.
ERROR java.sql.SQLException: Access denied for user 'root'@'50.16.35.xxx' (using password: NO)
In error massage its trying to connect my local machine again because my local machine IP address is 50.16.35.xxx instead of 54.35.xx.xxx
Can anyone please tell me how this is happening.
Plz help me : lakhveer.singh51@gmail.com
Reply

Nandhakumar Ravi says
DECEMBER 17, 2015 AT 8:05 AM
will jdk 1.8 works with oracle11g r2
Reply

Shank says
SEPTEMBER 14, 2014 AT 11:47 PM
Hello,
I want to call 4 database schema through a single jdbc connection. How can i do this thing??
Reply

Hayet says
JULY 31, 2014 AT 3:45 PM
Hello,
my program allows me to add to my DB a list of petri nets each one is characterized by a matrix i managed to create and add them using an array and stocking the values in a text field separated by "/" but i need to modify them and i don't know how to proceed i need to get the values from the DB and fill the labels so i can modify and register the new values.
can you please help me with that .
thank you
Reply

Rahul says

JUNE 10, 2014 AT 10:00 PM

When i am trying to insert the data into employee table from callable statement i am getting the following error:
java.sql.SQLException: ORA-06550: line 1, column 7:
PLS-00201: identifier 'INSERTEMPLOYEE' must be declared
ORA-06550: line 1, column 7:
PL/SQL: Statement ignored
I am using the same insert procedure insertEmployee you have given in this website…Please help me.

Reply

Pankaj says

JUNE 11, 2014 AT 9:18 AM

it looks like the issue with package, please check if it's compiled successfully.

Reply

sarang says

APRIL 3, 2014 AT 12:15 PM

I want to develop below functionality in java will u help me please??!!
1. Create a User class with instance variables to hold the following data
1. code – numeric
2. name – text
3. age – numeric
4. hobbies – collection of strings
5. date of birth – date
2. Create a subclass of the above User class with the name – RegisteredUser with instance variables to hold the following data
1. emailid – text
2. password – text
3. Write a SQL script to create 2 tables – app_users & app_registered_users to store data of the above 2 classes respectively.
4. Create a properties file with following properties
1. db.url – hold url of the database to connect to
2. db.driver – name of the database driver class
3. db.username – db username
4. db.password – db password
5. Create class DBConnectionManager with a the following
1. static variable of type java.util.Properties

2. static initializer block – Load the properties file created in point 4 above using the java.util.Properties class
3. static method getConnection with no parameters & java.sql.Connection return type.
The method should create & return connection to the DB.
6. Create a subclass – UserNotFoundException by extending RuntimeException class
7. Create a class UserDAO containing the following methods
1. boolean save(User user) throws Exception – this method should save the user object to the app_users table.
2. boolean save(RegisteredUser user) throws Exception – this method should save the user to the app_registered_users table.
2. List list() throws Exception – should list out all the users in the app_users table
3. User findByCode(Long code) throws Exception – find the user given his code, if no user is found with the given code, return null
4. void update(User user) throws Exception – update the user, All the fields except the code & name, should be updated
5. void delete(long code) throws Exception – delete the user with the given code, if user is not found, throw UserNotFoundException
Note: In each of these methods you will need to use the DBConnectionManager to get the connection to the DB.
8. Create a class UserManager as follows
1. static instance variable holding reference to UserManager class
2. static method with the signature – UserManager getInstance() – if the static instance variable is null, invoke the default constructor and return the static instance variable
3. create non static instance variable referencing – UserDAO
4. private default constructor – create new Object of UserDao and store in the non static instance variable declared in the above point
5. non static instance method – void createUser(String name, Date dob, String… hobbies) –
Create a unique number to be used a code to be used as users code
Create a object of User and store it to DB using the UserDao instance variable. You will also need to check that no 2 users have the same username, password & code
6. non static instance method – void createUser(String name, Date dob, String emailID, String password, List hobbies) –
create a unique number to be used as code to be used as users code
create a object of RegisteredUser class and store it to DB using the UserDao instance variable – The data should get stored in the app_registered_users table
7. non static instance method – void updateUser(long code, String name, Date dob, String… hobbies) – this should update an existing user
Reply

Pankaj says
APRIL 3, 2014 AT 3:43 PM
This looks like a complete project or assignment. Read the documents and you should do it yourself to gain more understanding.

Reply

sarang says
APRIL 4, 2014 AT 5:26 PM
yes i am doing it but I need some guidelines or approach to develop this like some steps if u can suggest thnx in adv!!
Reply

Hùng says
NOVEMBER 7, 2016 AT 7:54 PM
Yep, I think you should do it yourself to gain more understanding. So hard to understanding if you don't try on it.
Reply

Rishi Chopra says
JANUARY 19, 2014 AT 10:01 PM
Thanks Pankaj for great article. I am trying to resurrect my CV back and this helps a lot.
Reply

Madiraju Krishna Chaitanya says
JANUARY 10, 2014 AT 2:32 PM
Hi Pankaj Ji,Nice Article.Thank You.Please continue with your Great Work… □
Reply

Siddu says
JANUARY 7, 2014 AT 4:14 AM
Very Good Explanation, I have one qn on ResultSet Objects. Say suppose i have 10 records in my table aftter executing below statement. How selected records stored in java. Will it be like array of 10 records or some other way. Could you please explain.
ResultSet rs = stmt.executeQuery(QUERY))
Reply

Syed Talha says

SEPTEMBER 4, 2014 AT 12:22 AM

i have web service and using this approach to access DB in java.. but i dont want to get connection again and again… my objective is to open connection first time and every calls get same connection.. please help.

Thanks

Reply

Pankaj says

SEPTEMBER 4, 2014 AT 3:09 AM

Create connection pool using JNDI in your server and then connect using DataSource in your application. There are many posts for this here, use search to find out what you are looking for.

Reply

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment

Name *

Email *

Save my name, email, and website in this browser for the next time I comment.

POST COMMENT

- Java Logging
- Java Annotations
- Java XML
- Collections in Java
- Java Generics
- Exception Handling in Java
- Java Reflection
- Java Design Patterns
- JDBC Tutorial

Java EE Tutorials

- Servlet JSP Tutorial
- Struts2 Tutorial
- Spring Tutorial
- Hibernate Tutorial
- Primefaces Tutorial
- Apache Axis 2
- JAX-RS
- Memcached Tutorial