Introduction to JDBC:

- Java Database Connectivity
- JDBC is a Java API, which will connect to any kind of database and will execute respected query.
- JDBC API consists of JDBC drivers, in order to connect Project with database.
- With the help of JDBC driver, we can save, delete, update, retrieve data into / from database.
- All the classes related JDBC resides inside java.sql package.
- Functionality Of JDBC:
    (1) Connect to database
    (2) Update, Insert, Delete data into database.
    (3) Retrieve data from the database.
- Some of the frequently used classes of JDBC:
- 1) Connection → To connect our application to Database.
- 2) PreparedStatement → To execute statements like, Insert, Update and Delete. If they're executed correctly it will return 1 otherwise null.
- 3) ResultSet: It is used for select query, when you execute select query it returns rows. Those rows are stored in a ResultSet Object.

  We'll cover more about these later, while building project.

JDBC Drivers:

1) JDBC – ODBC Bridge Driver:

- Uses ODBC driver to connect database.
- The JDBC-ODBC driver converts JDBC method calls into the ODBC function calls.
- There are now thin drivers, they discouraged this thing of conversion.
- *Advantages:* Easy to use, can be easily connected to the database.
- Disadvantages: Performance degrades as the JDBC method calls are converted into ODBC function Calls. JDBC driver needs to be installed in client's machine.

2) The native API drivers:

- Uses client side libraries of the database.
- This driver converts JDBC calls into native calls of the database API, It's not written in Java.
- Advantage: Performance upgrades than JDBC-ODBC drivers
- Disadvantage: Native drivers needs to be installed on client machine and Vendor libraries needs to be installed in client machine.

3) **Network Protocol Driver:**

- NPD uses middleware that converts JDBC calls directly or indirectly into the vendor specific database control.
- It is fully written in Java.
- Advantage: No client side libraries are required because of Application server that performs many tasks like etc.
- Disadvantage: Network support is required on client's side.
- Database specific coding needs to be done.
- Maintenance is high.

4) **Thin Driver:**

- Converts JDBC calls directly into the vendor specific protocol
- Thats why it's called thing driver
- Fully written in Java
- **Advantage:** Better performance than any other driver.
- NO software required at client side.

How to connect Java Program to Database:

Step1: **Necessary Initial Working**

Firstly load JAR file in your project library and in program import **java.sql** package.

Step2: **Registering JDBC Driver:**

Class.forName method helps us to register JDBC class (from JAR file) into our program.

Example:

```
Class.forName("com.mysql.cj.jdbc.Driver");
Class.forName("oracle.jdbc.driver.OracleDriver");
```

Step 3: **Write URL string using getConnection() method:**

In order to connect our Database with our application we have to provide basic information about our database to our application, that is done using *getConnection()* method of *DriverManager* class.

Example:

```
Connection conn =
DriverManager.getConnection("jdbc:mysql://localhost
:3306/tushar", "root", "Tushar2408");
```

jdbc:mysql://localhost:3306 : It's a way of telling application that we're using MySQL database, which is at 3306 port and within I want to operate "tushar" database.

Root: It's a username of MySQL workbench.

Tushar2408: It's a password for that User.

You might get error because of unhandled exceptions, don't forgot to surround the block with Try-Catch.

Congrats, your application is now connected to a database, let's perform some operation now.

Step 4: **Insert Operation:**

If you remember we discussed, PreparedStatement will deal with Inserting, Deleting and Updation of data.

Declare PrepareStatement type of variable globally:

ex: PreparedStatement pst;

First let's see example then we will discuss how things work.

```
        String pname = jTextField1.getText();
        String price = jTextField2.getText();
        String qty = jTextField3.getText();

            pst = con.prepareStatement("insert into
product(pname, price,qty)values(?,?,?)");

            pst.setString(1, pname);
            pst.setString(2, price);
            pst.setString(3, qty);

            int k = pst.executeUpdate();
```

```
            if(k == 1){
                JOptionPane.showMessageDialog(this,
"Record Added");

            }
        }
```

We have 3 variables name, price, qty we want to send them into database.

Con: it's the connection variable, it contains basic information about Database. On which database we should perform, port number etc.

con has method called preparedStatement, it will contain basic SQL syntax, we all know that. But one question remains. Why did you use "?" at values.

We will now set values on those questionmarks with setString / setInt method. Please remember out con. PreparedStatement is stored at pst variable.

```
pst.setString(1, pname);
```

1 indicates the position of question mark, ex: please set “XYZ” product name in first question mark.

Pname indicates what value you want to store into that question mark.

Once query is ready, what you do? You execute them. We will do the same.

Query is executed, using → `pst.executeUpdate();`

If it’s successfully executed, it will return “1”, otherwise it’ll return respected error.

Now if “*k == 1*” We will assume, our query was executed successfully.

The same way you can perform, Update and Delete. Pretty Easy Right?

Step 5: **Retrieving Data from database.**

For retrieving, ResultSet comes to the picture.

Let's see example:

```
pst = con.prepareStatement("select * from
product where pid = 1");
```

We wrote query for retrieving the data of Product no 1, here query is ready. Now what will we do? We'll fire the query.

But the result returned by it, are the **ResultSet** type. So it must be stored inside **ResultSet** variable, correct?

```
ResultSet rs = pst.executeQuery();
```

The data with PID 1, is now stored inside rs variable.

How to access them into application?

```
if(rs.next()){
   jTextField1.setText(rs.getString(2))
    jTextField2.setText(rs.getString(3));
    jTextField3.setText(rs.getString(4));
         }
```

if there is data with PID 1 in our database, then rs.next() will be evaluated to true, otherwise false.

If it's true, **getString()** method is used to get value from **rs** variable.

GetString() contains one parameter and it is column number of the field you want to access.

NOTE: This is not a material, this document is just a reference for better understanding of topics. Thank you. I hope it helps you.