Basic Components of JDBC Application:

Driver: Java Calls←→DB Calls

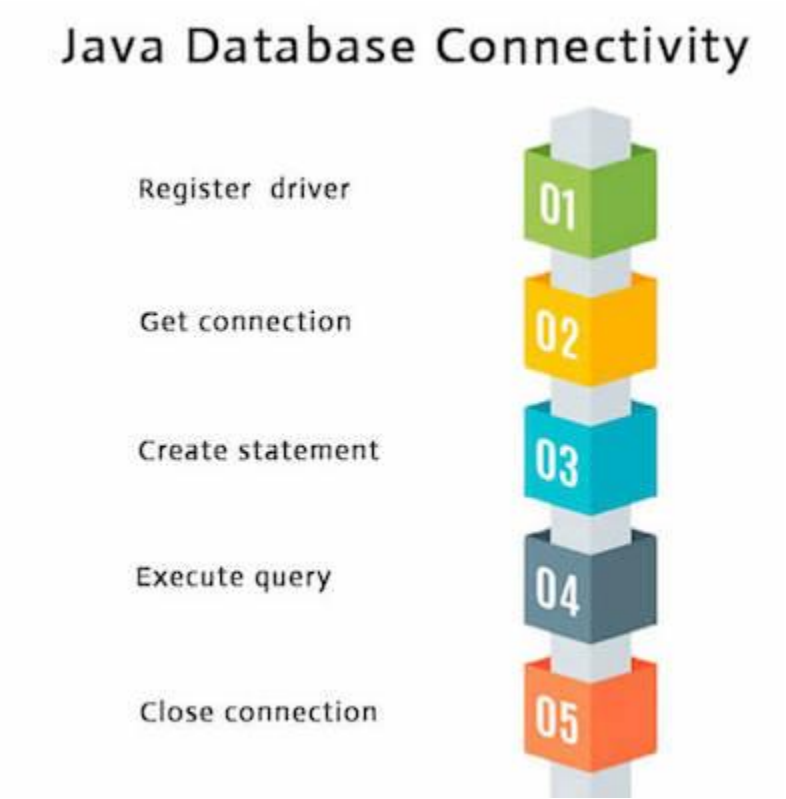
JAVA and Database are both different languages. To convert Java calls into DB calls and DB calls into Java calls, we require one translator; that translator is nothing but the Driver.

Connections: Network Socket

Statement Object: It is responsible for sending the Query to the DB, and the DB engine must execute the SQL query and bring the results from the DB to the Java application.

ResultSet: It holds the results of a SQL query; Java applications can get the results from the ResultSet.

5 Steps to connect to the database in Java



There are five steps to connect any Java application with the database in Java using JDBC. They are as follows:

- Register for the driver's class
- Creating a connection
- Creating a statement
- Executing queries
- Closing connection

1) Register for the driver class

The forName() method of class Class is used to register the driver class. This method is used to load the driver class dynamically.

Syntax of forName() method

```
1. public static void forName(String className) throws ClassNotFoundException
```

Example

```
1. Class.forName("com.mysql.jdbc.Driver ");
```

2) Create the connection object

The getConnection() method of DriverManager class is used to establish a 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 user_name,String password)  thr
   ows SQLException
```

Example to establish a connection with the Oracle database

```
Connection con = DriverManager.getConnection(url,user,pwd);
```

3) Create the Statement object

The createStatement() method of Connection interface is used to create a statement. The object of the statement is responsible to execute queries with the database.

Syntax of createStatement() method

```
1. public Statement createStatement()throws SQLException
```

Example to create the statement object

```
1. Statement stmt=con.createStatement();
```

4) Execute the query

The executeQuery() method of the Statement interface executes queries to the database. This method returns the object of ResultSet, which can be used to get all the table records.

Syntax of executeQuery() method

```
1. public ResultSet executeQuery(String sql) throws SQLException
```

Example to execute query

```
1. ResultSet rs=stmt.executeQuery("select * from emp");
2. while(rs.next()){
3. System.out.println(rs.getInt(1)+" "+rs.getString(2));
4. }
```

5) Close the connection object

By closing the connection object statement, the ResultSet will be closed automatically. The close() method of the Connection interface is used to close the connection.

Syntax of close() method

```
1. public void close()throws SQLException
```

Example to close connection

```
1. con.close();
```

Program: TestJDBC1.java

// We can insert another row in the database table using the following Java program.

```
import java.sql.*;
class TestJDBC1 {
   public static void main(String[] args) throws Exception {
Class.forName("com.mysql.jdbc.Driver").newInstance();
      Connection con =
DriverManager.getConnection("jdbc:mysql://localhost:3306/jdbc?user=root&password=saurabh
");
      Statement st = con.createStatement();
      String sql = "INSERT INTO emp(name,salary) VALUE('Sumit',32568.05)";
      st.executeUpdate(sql);
      st.close();
      con.close();
  System.out.println("---SQL executed successfully---");
   }
}
```

OR

```
import java.sql.*;
class TestJDBC1 {
   public static void main(String[] args) throws Exception {
```

/* **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. The driver class for the mysql database is com.mysql.jdbc.Driver.*/

```
Class.forName("com.mysql.jdbc.Driver").newInstance();
```

/* **2) Create the connection object:** The getConnection() method of DriverManager class is used to establish a connection with the database. The connection URL for the mysql database is jdbc:mysql://localhost:3306/jdbc 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 jdbc is the database name. We may use any database; in such case, we need to replace the jdbc with our database name.
Username: The default username for the mysql database is **root.**
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 saurabh as the password.*/

```
Connection con =
DriverManager.getConnection("jdbc:mysql://localhost:3306/jdbc?user=root&password =
saurabh");
```

/* 3) Create the Statement object: The createStatement() method of Connection interface is used to create statement. The object of the statement is responsible to execute queries with the database. public Statement createStatement(): creates a statement object that can be used to execute SQL queries.*/

```
Statement st = con.createStatement();
```

/* 4) Execute the query: The executeUpdate() method of the Statement interface is used to execute queries to the database. It is used to execute the specified query; it may be create, drop, insert, update, delete etc.*/

```
String sql = "INSERT INTO emp(name,salary) VALUE('Sumit',32568.05)";
st.executeUpdate(sql);
```

/* 5) Close the connection object: By closing the connection object statement will be closed automatically. The close() method of the Connection interface is used to close the connection.*/

```
st.close();
con.close();
System.out.println("---SQL executed successfully---");
  }
}
```

```
D:\1 Java\Programs>javac TestJDBC1.java

D:\1 Java\Programs>java TestJDBC1
---SQL executed successfully---
```

```
mysql> select * from emp;
+-------+--------+------------+
| empId | name   | salary     |
+-------+--------+------------+
|     1 | deepak |  75000.250 |
|     2 | rohan  |  65000.000 |
|     3 | aditi  |  87000.344 |
|     4 | Aman   | 256856.047 |
|     5 | Sumit  |  32568.051 |
+-------+--------+------------+
5 rows in set (0.00 sec)
```

Program: TestDelete.java

// We can delete a row from the database table using the following Java program.

```
import java.sql.*;
class TestDelete {
   public static void main(String[] args) throws Exception {
      Class.forName("com.mysql.jdbc.Driver").newInstance();
      Connection con =
DriverManager.getConnection("jdbc:mysql://localhost:3306/jdbc?user=root&password=saurabh
");
      Statement st = con.createStatement();
      String sql = "DELETE FROM emp where empId=1";
      st.executeUpdate(sql);
      st.close();
      con.close();
      System.out.println("---SQL executed successfully---");
   }
}
```

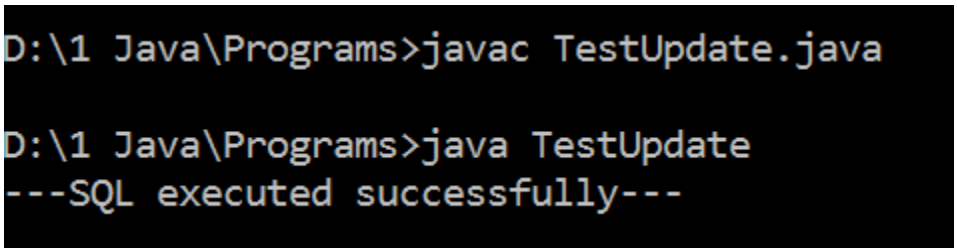
```
D:\1 Java\Programs>javac TestDelete.java

D:\1 Java\Programs>java TestDelete
---SQL executed successfully---
```

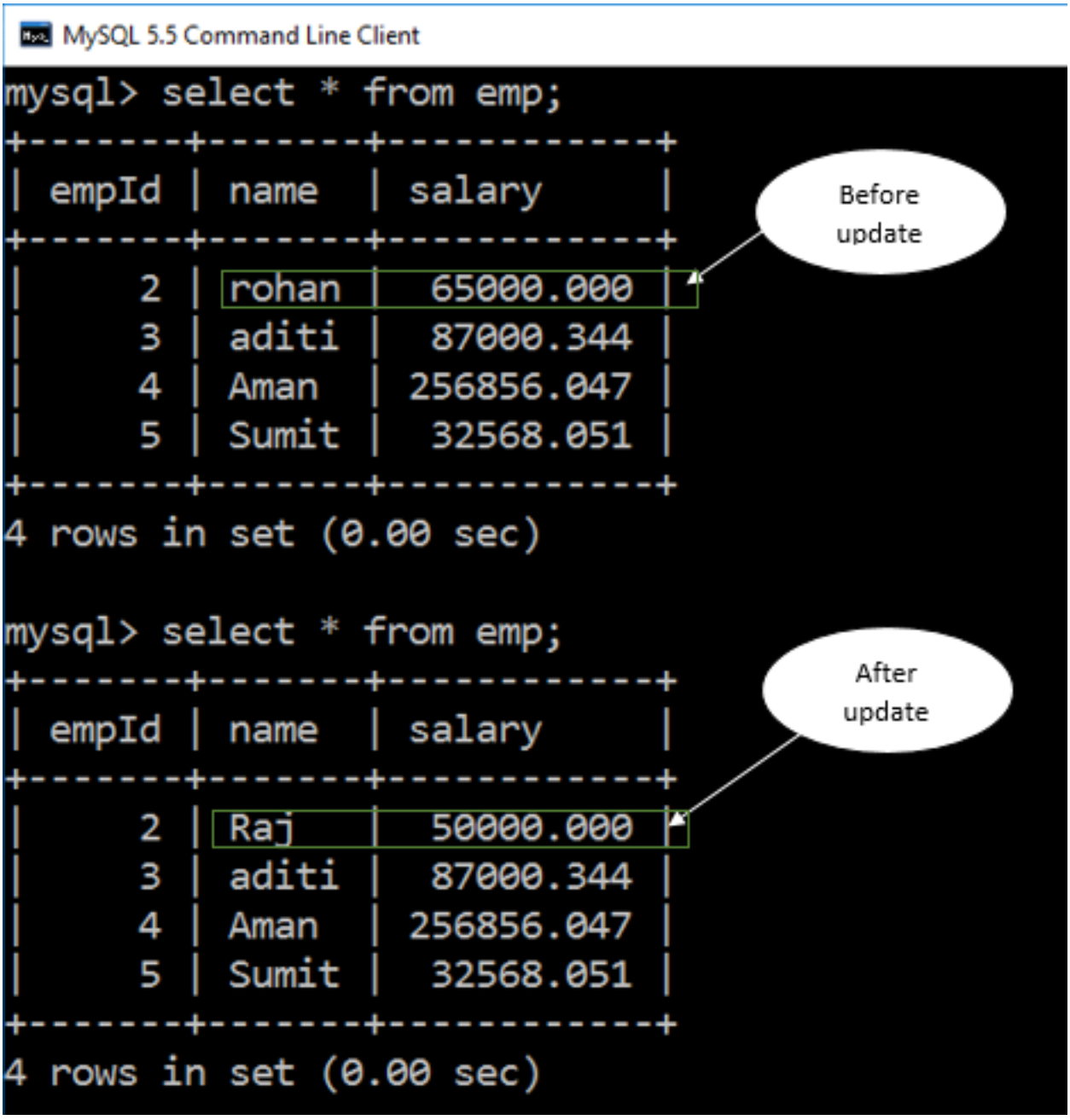
```
mysql> select * from emp;
+-------+-------+------------+
| empId | name  | salary     |
+-------+-------+------------+
|     2 | rohan |  65000.000 |
|     3 | aditi |  87000.344 |
|     4 | Aman  | 256856.047 |
|     5 | Sumit |  32568.051 |
+-------+-------+------------+
4 rows in set (0.00 sec)
```

Program: TestUpdate.java

//The following Java program can update the record in the database table.

```
import java.sql.*;
class TestUpdate {
   public static void main(String[] args) throws Exception {
      Class.forName("com.mysql.jdbc.Driver").newInstance();
      Connection con =
DriverManager.getConnection("jdbc:mysql://localhost:3306/jdbc?user=root&password=saurabh
");
      Statement st = con.createStatement();
      String sql = "UPDATE emp SET name='Raj', salary=50000 where empId=2";
      st.executeUpdate(sql);
     st.close();
      con.close();
      System.out.println("---SQL executed successfully---");
   }
}
```




MySQL 5.5 Command Line Client
mysql> select * from emp;
+-------+-------+------------+
| empId | name | salary |
+-------+-------+------------+
2	rohan	65000.000
3	aditi	87000.344
4	Aman	256856.047
5	Sumit	32568.051
+-------+-------+------------+
4 rows in set (0.00 sec)
Before update
mysql> select * from emp;
+-------+-------+------------+
| empId | name | salary |
+-------+-------+------------+
2	Raj	50000.000
3	aditi	87000.344
4	Aman	256856.047
5	Sumit	32568.051
+-------+-------+------------+
4 rows in set (0.00 sec)
After update

PreparedStatement Interface

- The PreparedStatement interface is a sub-interface of Statement.
- It is used to execute a parameterized query (? Parameter).
- It handles SQL Injection Problems.
- Reuse parser object for similar queries
- Faster than Statement
- The application's performance will be faster using the PreparedStatement interface because the query is compiled only once.

  Ex: String sql="insert into emp values(?,?,?)";

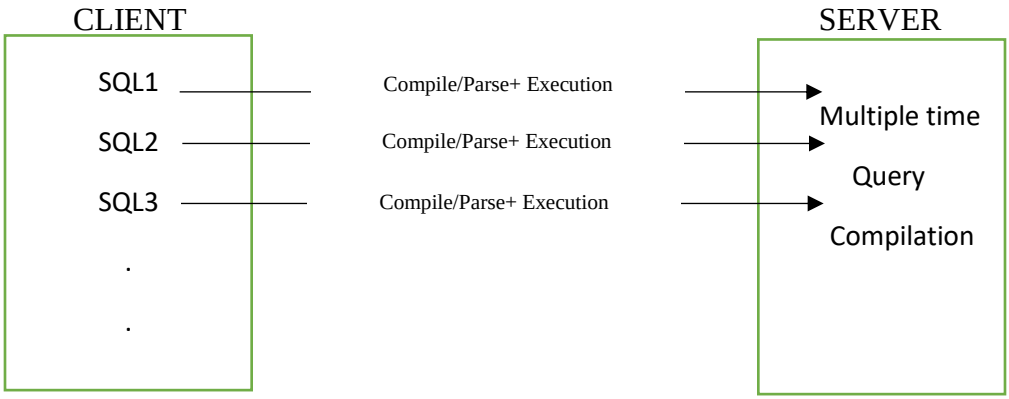


FIG: USING STATEMENT INTERFACE

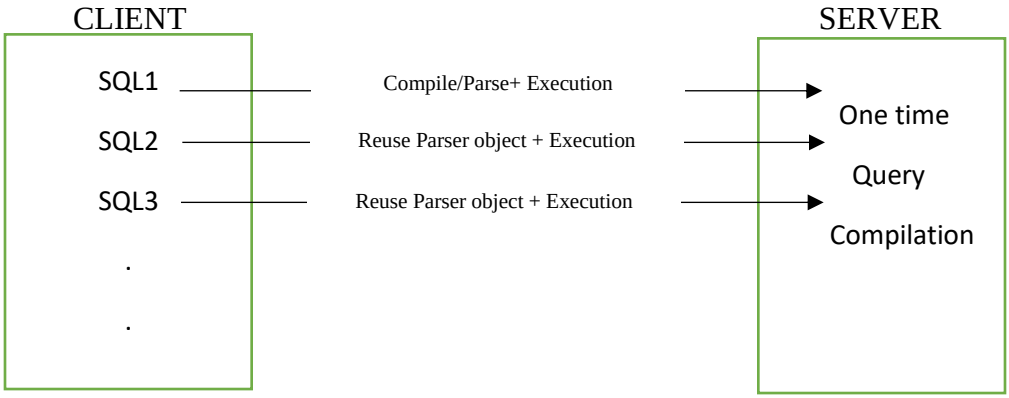


FIG: USING PREPAREMENT STATEMENT INTERFACE

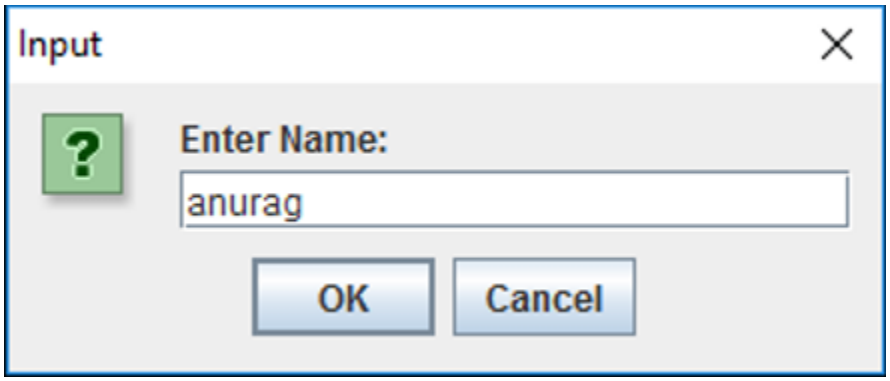
If SQL1, SQL2, SQL3… are same queries with different data. So, it doesn't need to be parsed/compiled multiple times; it needs to be parsed/compiled only once with different data/datasets.

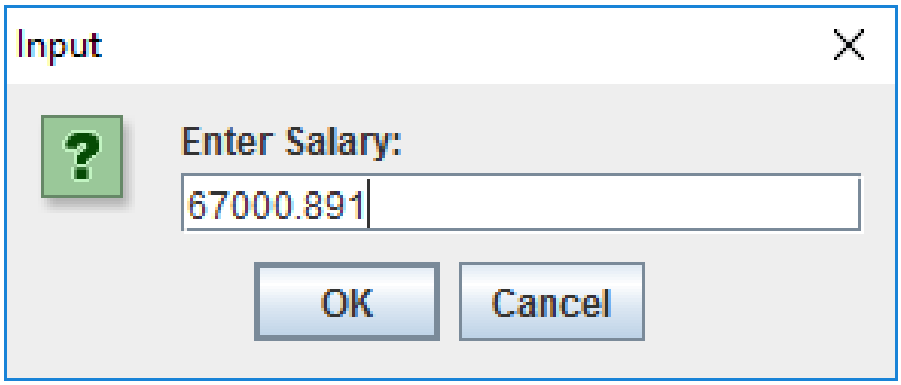
DynamicInsertPST.java

This program takes input (name and salary) from the user at runtime using a dialog box (JOptionPane) and inserts it into a MySQL database table named emp using PreparedStatement.

```
import java.sql.*;
import javax.swing.JOptionPane;
class DynamicInsertPST {
   public static void main(String[] args) throws Exception {
      Class.forName("com.mysql.jdbc.Driver").newInstance();
      Connection con =
DriverManager.getConnection("jdbc:mysql://localhost:3306/jdbc?user=root&password=saurabh
");
      String n=JOptionPane.showInputDialog("Enter Name:");
      String s=JOptionPane.showInputDialog("Enter Salary:");
      float fs=Float.parseFloat(s);

      // ? : place holder or parameter (parameterized query)
      String sql = "INSERT INTO emp(name,salary) VALUE(?,?)";
      PreparedStatement st = con.prepareStatement(sql);
      //bind data in PST
      st.setString(1, n);// 1 specifies the first parameter in the query
      st.setFloat(2, fs);  // 2  specifies the second parameter in the query
      st.executeUpdate(); //no arguments
      st.close();
      con.close();
      System.out.println("---SQL executed successfully---");
   }
}
```

```
D:\1 Java\Programs>java DynamicInsertPST
---SQL executed successfully---
```

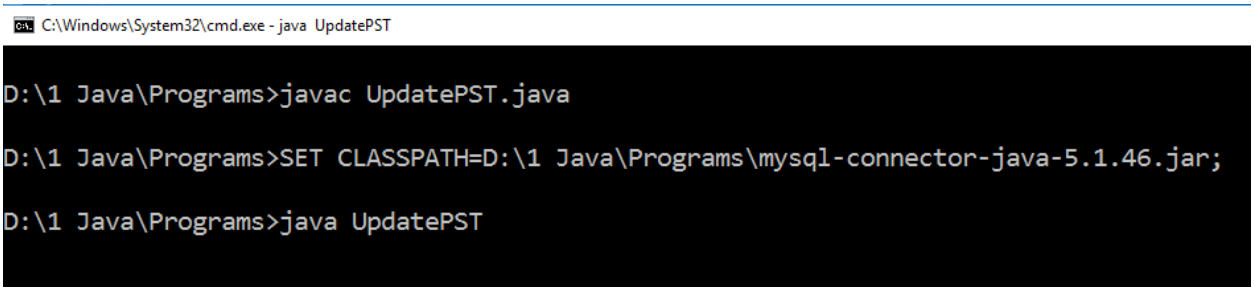


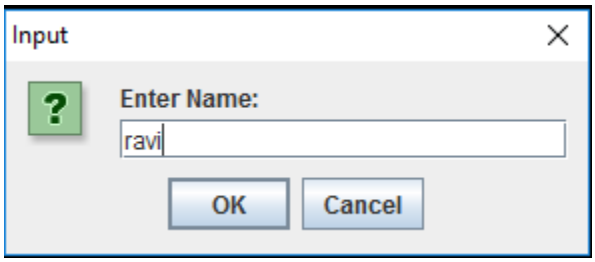


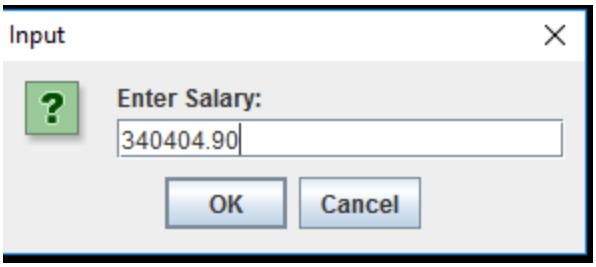

```
mysql> select * from emp;
+-------+--------+------------+
| empId | name   | salary     |
+-------+--------+------------+
|     2 | Raj    |  50000.000 |
|     3 | aditi  |  87000.344 |
|     4 | Aman   | 256856.047 |
|     5 | Sumit  |  32568.051 |
|     6 | anurag |  67000.891 |
+-------+--------+------------+
5 rows in set (0.06 sec)
```

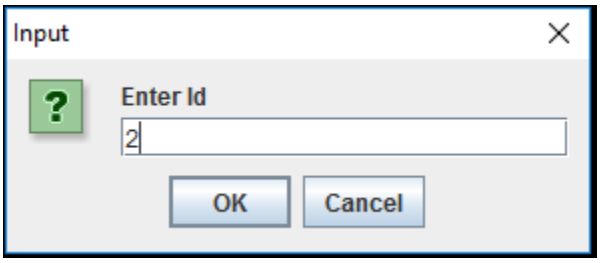
UpdatePST.java

```
import java.sql.*;
import javax.swing.JOptionPane;
class UpdatePST {
   public static void main(String[] args) throws Exception {
      Class.forName("com.mysql.jdbc.Driver").newInstance();
      Connection con =
DriverManager.getConnection("jdbc:mysql://localhost:3306/jdbc?user=root&password=saurabh
");
     String n=JOptionPane.showInputDialog("Enter Name:");
      String s=JOptionPane.showInputDialog("Enter Salary:");
      float fs=Float.parseFloat(s);
       String sid=JOptionPane.showInputDialog("Enter Id");
      int id =Integer.parseInt(sid);
      String sql = "UPDATE emp SET name=?,salary=? WHERE empId=?";
      PreparedStatement st = con.prepareStatement(sql);
      st.setString(1, n);
      st.setFloat(2, fs);
      st.setInt(3, id); //PK
      st.executeUpdate();
      st.close();
      con.close();
      System.out.println("---SQL executed successfully---");
   }
}
```






Input
Enter Salary:
340404.90
OK
Cancel


Input
Enter Id
2
OK
Cancel

```
mysql> select * from emp;
+-------+--------+------------+
| empId | name   | salary     |
+-------+--------+------------+
|     2 | Raj    |  50000.000 |
|     3 | aditi  |  87000.344 |
|     4 | Aman   | 256856.047 |
|     5 | Sumit  |  32568.051 |
|     6 | anurag |  67000.891 |
+-------+--------+------------+
5 rows in set (0.00 sec)

mysql> select * from emp;
+-------+--------+------------+
| empId | name   | salary     |
+-------+--------+------------+
|     2 | ravi   | 340404.906 |
|     3 | aditi  |  87000.344 |
|     4 | Aman   | 256856.047 |
|     5 | Sumit  |  32568.051 |
|     6 | anurag |  67000.891 |
+-------+--------+------------+
5 rows in set (0.00 sec)
```

ResultSet Interface

In Java, the ResultSet is the Object that holds the result of a database query, typically the SQL select statement. It is also part of the JDBC API used to interact with relational databases. The ResultSet allows us to traverse the rows of tables returned by the SQL query and extract a specific column from the SQL query result.

The ResultSet is essentially a table of data where each row represents a record, and each column represents a field in the database. The ResultSet has a cursor that points to the current row in the ResultSet and we can be able to navigate in ResultSet by using the **next(), previous(), first(),** and **last()** methods**.** We can retrieve data by using different methods like **getString(), getInt(), getDouble()** and other methods.

The object of ResultSet maintains a cursor pointing to a table row. Initially, the cursor points to before the first row.

The program fetches and displays employee records (empId, name, salary) from the emp table, where the employee ID (empId) is less than or equal to a given value (in this case, 6).

```
import java.sql.*;
class RS {
   public static void main(String[] args) throws Exception {
      Class.forName("com.mysql.jdbc.Driver").newInstance();
      Connection con =
DriverManager.getConnection("jdbc:mysql://localhost:3306/jdbc?user=root&password=saurabh
");

      String sql = "SELECT empId, name ,salary FROM emp WHERE empId <= ?";
      PreparedStatement st = con.prepareStatement(sql);
     // st.setInt(1, 3);//prints data where empId <= 3
        st.setInt(1, 6);//// prints data where empId <= 6

ResultSet rs = st.executeQuery();//Executes the query and stores the result set (records) in rs.

      while (rs.next()) {
         System.out.println(rs.getInt("empId")+" "+rs.getString("name")+ "
"+rs.getFloat("salary"));
      }
      rs.close();
      st.close();
      con.close();
      System.out.println("---SQL executed successfully---");
   }
}
```

```
D:\1 Java\Programs>javac RS.java

D:\1 Java\Programs>java RS
2 ravi 340404.9
3 aditi 87000.34
4 Aman 256856.05
5 Sumit 32568.05
6 anurag 67000.89
---SQL executed successfully---

D:\1 Java\Programs>javac RS.java

D:\1 Java\Programs>java RS
2 ravi 340404.9
3 aditi 87000.34
---SQL executed successfully---
```

A list of popular *interfaces* of JDBC API is given below:

- Driver interface
- Connection interface
- Statement interface
- PreparedStatement interface
- CallableStatement interface
- ResultSet interface
- ResultSetMetaData interface
- DatabaseMetaData interface
- RowSet interface

A list of popular classes of JDBC API is given below:

- DriverManager class
- Blob class
- Clob class
- Types class

Assignment Topics:

- CallableStatement interface
- RowSet interface