Spring JdbcTemplate Tutorial
-Refer to https://www.javatpoint.com/spring-JdbcTemplate-tutorial
-https://docs.spring.io/spring/docs/4.0.x/spring-framework-reference/html/jdbc.html
1. Spring JdbcTemplate is a powerful mechanism to connect to the database and execute SQL queries.

2. It internally uses JDBC api, but eliminates a lot of problems of JDBC API.

3. Problems of JDBC API
1)We need to write a lot of code before and after executing the query, such as creating connection, statement, closing resultset, connection etc.
2)We need to perform exception handling code on the database logic.
3)We need to handle transaction.
4)Repetition of all these codes from one to another database logic is a time consuming task.

4. Advantage of Spring JdbcTemplate
1)Spring JdbcTemplate eliminates all the above mentioned problems of JDBC API.
2)It provides you methods to write the queries directly, so it saves a lot of work and time.

5. Spring Jdbc Approaches
-Spring framework provides following approaches for JDBC database access:
1)JdbcTemplate
2)NamedParameterJdbcTemplate
3)SimpleJdbcTemplate
4)SimpleJdbcInsert and SimpleJdbcCall

6. JdbcTemplate class
1)It is the central class in the Spring JDBC support classes.
2)It takes care of creation and release of resources such as creating and closing of connection object etc.
3)So it will not lead to any problem if you forget to close the connection.
4)It handles the exception and provides the informative exception messages by the help of excepion classes defined in the org.springframework.dao package.
5)We can perform all the database operations by the help of JdbcTemplate class such as insertion, updation, deletion and retrieval of the data from the database.

7. The methods of spring JdbcTemplate class.

Method	Description
public int update(String query)	is used to insert, update and delete records.
public int update(String query,Object... args)	is used to insert, update and delete records using PreparedStatement using given arguments.
public void execute(String query)	is used to execute DDL query.
public T execute(String sql, PreparedStatementCallback action)	executes the query by using PreparedStatement callback.
public T query(String sql, ResultSetExtractor rse)	is used to fetch records using ResultSetExtractor.
public List query(String sql, RowMapper rse)	is used to fetch records using RowMapper.

8. Example of Spring JdbcTemplate

1)Create Table

```
create table employee(
  id number(10),
  name varchar2(100),
  salary number(10)
);
```

2)Employee.java

```
package com.example;

public class Employee {
  private int id;
  private String name;
  private float salary;
  //no-arg and parameterized constructors
  //getters and setters
}
```

3)EmployeeDao.java

```
package com.example;
import org.springframework.jdbc.core.JdbcTemplate;

public class EmployeeDao {
  private JdbcTemplate jdbcTemplate;

  public void setJdbcTemplate(JdbcTemplate jdbcTemplate) {
    this.jdbcTemplate = jdbcTemplate;
  }

  public int saveEmployee(Employee e){
    String query="insert into employee values(
    '"+e.getId()+"','"+e.getName()+"','"+e.getSalary()+"')";
    return jdbcTemplate.update(query);
  }
  public int updateEmployee(Employee e){
    String query="update employee set
    name='"+e.getName()+"',salary='"+e.getSalary()+"' where id='"+e.getId()+"' ";
    return jdbcTemplate.update(query);
  }
  public int deleteEmployee(Employee e){
    String query="delete from employee where id='"+e.getId()+"' ";
    return jdbcTemplate.update(query);
  }
}
```

4)applicationContext.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans
  xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:p="http://www.springframework.org/schema/p"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
```

```
 99          http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
100
101       <bean id="ds" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
102         <property name="driverClassName" value="oracle.jdbc.driver.OracleDriver" />
103         <property name="url" value="jdbc:oracle:thin:@localhost:1521:xe" />
104         <property name="username" value="system" />
105         <property name="password" value="oracle" />
106       </bean>
107
108       <bean id="jdbcTemplate" class="org.springframework.jdbc.core.JdbcTemplate">
109         <property name="dataSource" ref="ds"></property>
110       </bean>
111
112       <bean id="edao" class="com.example.EmployeeDao">
113         <property name="jdbcTemplate" ref="jdbcTemplate"></property>
114       </bean>
115
116     </beans>
117
118   5)Test.java
119     package com.example;
120
121     import org.springframework.context.ApplicationContext;
122     import org.springframework.context.support.ClassPathXmlApplicationContext;
123     public class Test {
124
125       public static void main(String[] args) {
126         ApplicationContext ctx=new ClassPathXmlApplicationContext("applicationContext.xml");
127
128         EmployeeDao dao=(EmployeeDao)ctx.getBean("edao");
129         int status=dao.saveEmployee(new Employee(102,"Amit",35000));
130         System.out.println(status);
131
132         /*int status=dao.updateEmployee(new Employee(102,"Sonoo",15000));
133           System.out.println(status);
134         */
135
136         /*Employee e=new Employee();
137         e.setId(102);
138         int status=dao.deleteEmployee(e);
139         System.out.println(status);*/
140       }
141     }
142
143
144 9. Example of PreparedStatement in Spring JdbcTemplate
145   1)We can execute parameterized query using Spring JdbcTemplate by the help of execute()
      method of JdbcTemplate class.
146   2)To use parameterized query, we pass the instance of PreparedStatementCallback in the
      execute method.
147   3)Syntax of execute method to use parameterized query
148     public T execute(String sql,PreparedStatementCallback<T>);
149
150   4)PreparedStatementCallback interface
```

```
   -It processes the input parameters and output results.
   -In such case, you don't need to care about single and double quotes.

5)Method of PreparedStatementCallback interface
   -It has only one method doInPreparedStatement.
   -Syntax of the method is given below:

      public T doInPreparedStatement(PreparedStatement ps)throws SQLException,
      DataAccessException

6)Example of using PreparedStatement in Spring
7)Create Table
   create table employee(
      id number(10),
      name varchar2(100),
      salary number(10)
   );

8)Employee.java
   package com.example;

   public class Employee {
      private int id;
      private String name;
      private float salary;
      //no-arg and parameterized constructors
      //getters and setters
   }

9)EmployeeDao.java
   package com.example;
   import java.sql.PreparedStatement;
   import java.sql.SQLException;

   import org.springframework.dao.DataAccessException;
   import org.springframework.jdbc.core.JdbcTemplate;
   import org.springframework.jdbc.core.PreparedStatementCallback;

   public class EmployeeDao {
      private JdbcTemplate jdbcTemplate;

      public void setJdbcTemplate(JdbcTemplate jdbcTemplate) {
         this.jdbcTemplate = jdbcTemplate;
      }

      public Boolean saveEmployeeByPreparedStatement(final Employee e){
         String query="insert into employee values(?,?,?)";
         return jdbcTemplate.execute(query,new PreparedStatementCallback<Boolean>(){
            @Override
            public Boolean doInPreparedStatement(PreparedStatement ps)
                  throws SQLException, DataAccessException {

               ps.setInt(1,e.getId());
               ps.setString(2,e.getName());
```

```
            ps.setFloat(3,e.getSalary());

            return ps.execute();

          }
        });
      }
    }

  10)applicationContext.xml
    <?xml version="1.0" encoding="UTF-8"?>
    <beans
      xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:p="http://www.springframework.org/schema/p"
      xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

      <bean id="ds" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
        <property name="driverClassName" value="oracle.jdbc.driver.OracleDriver" />
        <property name="url" value="jdbc:oracle:thin:@localhost:1521:xe" />
        <property name="username" value="system" />
        <property name="password" value="oracle" />
      </bean>

      <bean id="jdbcTemplate" class="org.springframework.jdbc.core.JdbcTemplate">
        <property name="dataSource" ref="ds"></property>
      </bean>

      <bean id="edao" class="com.example.EmployeeDao">
        <property name="jdbcTemplate" ref="jdbcTemplate"></property>
      </bean>
    </beans>

  11)Test.java
    package com.example;

    import org.springframework.context.ApplicationContext;
    import org.springframework.context.support.ClassPathXmlApplicationContext;
    public class Test {

      public static void main(String[] args) {
        ApplicationContext ctx=new ClassPathXmlApplicationContext("applicationContext.xml");

        EmployeeDao dao=(EmployeeDao)ctx.getBean("edao");
        dao.saveEmployeeByPreparedStatement(new Employee(108,"Amit",35000));
      }
    }


10. ResultSetExtractor Example | Fetching Records by Spring JdbcTemplate
  1)We can easily fetch the records from the database using query() method of JdbcTemplate
  class where we need to pass the instance of ResultSetExtractor.
  2)Syntax of query method using ResultSetExtractor
```

```
257     public T query(String sql,ResultSetExtractor<T> rse)
258   3)ResultSetExtractor Interface
259     -ResultSetExtractor interface can be used to fetch records from the database.
260     -It accepts a ResultSet and returns the list.
261
262   4)Method of ResultSetExtractor interface
263     -It defines only one method extractData that accepts ResultSet instance as a parameter.
264     -Syntax of the method is given below:
265       public T extractData(ResultSet rs)throws SQLException,DataAccessException
266
267   5)Example of ResultSetExtractor Interface to show all the records of the table
268   6)Create Table
269     create table employee(
270       id number(10),
271       name varchar2(100),
272       salary number(10)
273     );
274
275   7)Employee.java
276     package com.example;
277
278     public class Employee {
279       private int id;
280       private String name;
281       private float salary;
282       //no-arg and parameterized constructors
283       //getters and setters
284       public String toString(){
285         return id+" "+name+" "+salary;
286       }
287     }
288
289   8)EmployeeDao.java
290     package com.example;
291     import java.sql.ResultSet;
292     import java.sql.SQLException;
293     import java.util.ArrayList;
294     import java.util.List;
295     import org.springframework.dao.DataAccessException;
296     import org.springframework.jdbc.core.JdbcTemplate;
297     import org.springframework.jdbc.core.ResultSetExtractor;
298
299     public class EmployeeDao {
300       private JdbcTemplate template;
301
302       public void setTemplate(JdbcTemplate template) {
303         this.template = template;
304       }
305
306       public List<Employee> getAllEmployees(){
307        return template.query("select * from employee",new
           ResultSetExtractor<List<Employee>>(){
308          @Override
309           public List<Employee> extractData(ResultSet rs) throws SQLException,
```

```
            DataAccessException {

          List<Employee> list=new ArrayList<Employee>();
          while(rs.next()){
            Employee e=new Employee();
            e.setId(rs.getInt(1));
            e.setName(rs.getString(2));
            e.setSalary(rs.getInt(3));
            list.add(e);
          }
          return list;
        }
        });
       }
     }
   }

9)applicationContext.xml
   <?xml version="1.0" encoding="UTF-8"?>
   <beans
     xmlns="http://www.springframework.org/schema/beans"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xmlns:p="http://www.springframework.org/schema/p"
     xsi:schemaLocation="http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

     <bean id="ds" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
       <property name="driverClassName" value="oracle.jdbc.driver.OracleDriver" />
       <property name="url" value="jdbc:oracle:thin:@localhost:1521:xe" />
       <property name="username" value="system" />
       <property name="password" value="oracle" />
     </bean>

     <bean id="jdbcTemplate" class="org.springframework.jdbc.core.JdbcTemplate">
       <property name="dataSource" ref="ds"></property>
     </bean>

     <bean id="edao" class="com.example.EmployeeDao">
       <property name="jdbcTemplate" ref="jdbcTemplate"></property>
     </bean>

   </beans>

10)Test.java
   package com.example;

   import java.util.List;

   import org.springframework.context.ApplicationContext;
   import org.springframework.context.support.ClassPathXmlApplicationContext;
   public class Test {

     public static void main(String[] args) {
       ApplicationContext ctx=new ClassPathXmlApplicationContext("applicationContext.xml");
```

```
364         EmployeeDao dao=(EmployeeDao)ctx.getBean("edao");
365         List<Employee> list=dao.getAllEmployees();
366
367         for(Employee e:list)
368           System.out.println(e);
369
370       }
371     }
372
373
374 11. RowMapper Example | Fetching records by Spring JdbcTemplate
375   1)Like ResultSetExtractor, we can use RowMapper interface to fetch the records from the
      database using query() method of JdbcTemplate class.
376   2)In the execute of we need to pass the instance of RowMapper now.
377   3)Syntax of query method using RowMapper
378     public T query(String sql,RowMapper<T> rm)
379
380   4)RowMapper Interface
381     -RowMapper interface allows to map a row of the relations with the instance of user-defined
      class.
382     -It iterates the ResultSet internally and adds it into the collection.
383     -So we don't need to write a lot of code to fetch the records as ResultSetExtractor.
384
385   5)Advantage of RowMapper over ResultSetExtractor
386     -RowMapper saves a lot of code becuase it internally adds the data of ResultSet into the
      collection.
387
388   6)Method of RowMapper interface
389     -It defines only one method mapRow that accepts ResultSet instance and int as the parameter
      list.
390     -Syntax of the method is given below:
391
392     public T mapRow(ResultSet rs, int rowNumber)throws SQLException
393
394   7)Example of RowMapper Interface to show all the records of the table
395   8)Create Table
396     create table employee(
397       id number(10),
398       name varchar2(100),
399       salary number(10)
400     );
401
402   9)Employee.java
403     package com.example;
404
405     public class Employee {
406       private int id;
407       private String name;
408       private float salary;
409       //no-arg and parameterized constructors
410       //getters and setters
411       public String toString(){
412         return id+" "+name+" "+salary;
413       }
```

```
414     }
415
416   8)EmployeeDao.java
417     package com.example;
418     import java.sql.ResultSet;
419     import java.sql.SQLException;
420     import java.util.ArrayList;
421     import java.util.List;
422     import org.springframework.dao.DataAccessException;
423     import org.springframework.jdbc.core.JdbcTemplate;
424     import org.springframework.jdbc.core.ResultSetExtractor;
425
426     public class EmployeeDao {
427       private JdbcTemplate template;
428
429       public void setTemplate(JdbcTemplate template) {
430         this.template = template;
431       }
432
433       public List<Employee> getAllEmployees(){
434         return template.query("select * from employee",new RowMapper<Employee>(){
435           @Override
436           public Employee mapRow(ResultSet rs, int rownumber) throws SQLException {
437             Employee e=new Employee();
438             e.setId(rs.getInt(1));
439             e.setName(rs.getString(2));
440             e.setSalary(rs.getInt(3));
441             return e;
442           }
443         });
444       }
445     }
446
447   9)applicationContext.xml
448     <?xml version="1.0" encoding="UTF-8"?>
449     <beans
450       xmlns="http://www.springframework.org/schema/beans"
451       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
452       xmlns:p="http://www.springframework.org/schema/p"
453       xsi:schemaLocation="http://www.springframework.org/schema/beans
454         http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
455
456       <bean id="ds" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
457         <property name="driverClassName" value="oracle.jdbc.driver.OracleDriver" />
458         <property name="url" value="jdbc:oracle:thin:@localhost:1521:xe" />
459         <property name="username" value="system" />
460         <property name="password" value="oracle" />
461       </bean>
462
463       <bean id="jdbcTemplate" class="org.springframework.jdbc.core.JdbcTemplate">
464         <property name="dataSource" ref="ds"></property>
465       </bean>
466
467       <bean id="edao" class="com.example.EmployeeDao">
```

```
468         <property name="jdbcTemplate" ref="jdbcTemplate"></property>
469       </bean>
470
471     </beans>
472
473   10)Test.java
474     package com.example;
475
476     import java.util.List;
477
478     import org.springframework.context.ApplicationContext;
479     import org.springframework.context.support.ClassPathXmlApplicationContext;
480     public class Test {
481
482       public static void main(String[] args) {
483         ApplicationContext ctx=new ClassPathXmlApplicationContext("applicationContext.xml");
484         EmployeeDao dao=(EmployeeDao)ctx.getBean("edao");
485         List<Employee> list=dao.getAllEmployeesRowMapper();
486
487         for(Employee e:list)
488           System.out.println(e);
489
490       }
491     }
492
493
494 12. Spring NamedParameterJdbcTemplate Example
495   1)Spring provides another way to insert data by named parameter.
496   2)In such way, we use names instead of ?(question mark).
497   3)So it is better to remember the data for the column.
498   4)Simple example of named parameter query
499     insert into employee values (:id,:name,:salary)
500   5)Method of NamedParameterJdbcTemplate class
501     -In this example,we are going to call only the execute method of
      NamedParameterJdbcTemplate class.
502     -Syntax of the method is as follows:
503
504     pubic T execute(String sql,Map map,PreparedStatementCallback psc)
505
506   6)Example of NamedParameterJdbcTemplate class
507   7)Create Table
508     create table employee(
509       id number(10),
510       name varchar2(100),
511       salary number(10)
512     );
513
514   8)Employee.java
515     package com.example;
516
517     public class Employee {
518       private int id;
519       private String name;
520       private float salary;
```

```
521         //no-arg and parameterized constructors
522         //getters and setters
523       }
524
525     9)EmployeeDao.java
526       package com.example;
527
528       import java.sql.PreparedStatement;
529       import java.sql.SQLException;
530       import org.springframework.dao.DataAccessException;
531       import org.springframework.jdbc.core.PreparedStatementCallback;
532       import org.springframework.jdbc.core.namedparam.NamedParameterJdbcTemplate;
533       import java.util.*;
534
535       public class EmpDao {
536         NamedParameterJdbcTemplate template;
537
538         public EmpDao(NamedParameterJdbcTemplate template) {
539           this.template = template;
540         }
541         public  void save (Emp e){
542           String query="insert into employee values (:id,:name,:salary)";
543
544           Map<String,Object> map=new HashMap<String,Object>();
545           map.put("id",e.getId());
546           map.put("name",e.getName());
547           map.put("salary",e.getSalary());
548
549           template.execute(query,map,new PreparedStatementCallback() {
550             @Override
551             public Object doInPreparedStatement(PreparedStatement ps)
552                 throws SQLException, DataAccessException {
553               return ps.executeUpdate();
554             }
555           });
556         }
557       }
558
559     10)applicationContext.xml
560       <?xml version="1.0" encoding="UTF-8"?>
561       <beans
562         xmlns="http://www.springframework.org/schema/beans"
563         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
564         xmlns:p="http://www.springframework.org/schema/p"
565         xsi:schemaLocation="http://www.springframework.org/schema/beans
566           http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
567
568         <bean id="ds" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
569           <property name="driverClassName" value="oracle.jdbc.driver.OracleDriver" />
570           <property name="url" value="jdbc:oracle:thin:@localhost:1521:xe" />
571           <property name="username" value="system" />
572           <property name="password" value="oracle" />
573         </bean>
574
```

```
575         <bean id="jtemplate"
            class="org.springframework.jdbc.core.namedparam.NamedParameterJdbcTemplate">
576           <constructor-arg ref="ds"></constructor-arg>
577         </bean>
578
579         <bean id="edao" class="com.example.EmpDao">
580           <constructor-arg>
581             <ref bean="jtemplate"/>
582           </constructor-arg>
583         </bean>
584
585       </beans>
586
587     11)Test.java
588       package com.example;
589
590       import org.springframework.beans.factory.BeanFactory;
591       import org.springframework.beans.factory.xml.XmlBeanFactory;
592       import org.springframework.core.io.ClassPathResource;
593       import org.springframework.core.io.Resource;
594
595       public class SimpleTest {
596         public static void main(String[] args) {
597
598           Resource r=new ClassPathResource("applicationContext.xml");
599           BeanFactory factory=new XmlBeanFactory(r);
600
601           EmpDao dao=(EmpDao)factory.getBean("edao");
602           dao.save(new Emp(23,"sonoo",50000));
603
604         }
605       }
606
607
608   13. Spring SimpleJdbcTemplate Example
609     1)Spring 3 JDBC supports the java 5 feature var-args (variable argument) and autoboxing by
        the help of SimpleJdbcTemplate class.
610     2)SimpleJdbcTemplate class wraps the JdbcTemplate class and provides the update method
        where we can pass arbitrary number of arguments.
611
612     3)Syntax of update method of SimpleJdbcTemplate class
613       int update(String sql,Object... parameters)
614     4)We should pass the parameter values in the update method in the order they are defined in
        the parameterized query.
615     5)Example of SimpleJdbcTemplate class
616     6)Create Table
617       create table employee(
618         id number(10),
619         name varchar2(100),
620         salary number(10)
621       );
622
623     7)Employee.java
624       package com.example;
```

```
625
626     public class Employee {
627       private int id;
628       private String name;
629       private float salary;
630       //no-arg and parameterized constructors
631       //getters and setters
632     }
633
634   8)EmployeeDao.java
635     package com.example;
636
637     import org.springframework.jdbc.core.simple.SimpleJdbcTemplate;
638     public class EmpDao {
639       SimpleJdbcTemplate template;
640
641       public EmpDao(SimpleJdbcTemplate template) {
642         this.template = template;
643       }
644       public int update (Emp e){
645         String query="update employee set name=? where id=?";
646         return template.update(query,e.getName(),e.getId());
647
648         //String query="update employee set name=?,salary=? where id=?";
649         //return template.update(query,e.getName(),e.getSalary(),e.getId());
650       }
651     }
652
653   9)applicationContext.xml
654     <?xml version="1.0" encoding="UTF-8"?>
655     <beans
656       xmlns="http://www.springframework.org/schema/beans"
657       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
658       xmlns:p="http://www.springframework.org/schema/p"
659       xsi:schemaLocation="http://www.springframework.org/schema/beans
660         http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
661
662       <bean id="ds" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
663         <property name="driverClassName" value="oracle.jdbc.driver.OracleDriver" />
664         <property name="url" value="jdbc:oracle:thin:@localhost:1521:xe" />
665         <property name="username" value="system" />
666         <property name="password" value="oracle" />
667       </bean>
668
669       <bean id="jtemplate" class="org.springframework.jdbc.core.simple.SimpleJdbcTemplate">
670         <constructor-arg ref="ds"></constructor-arg>
671       </bean>
672
673       <bean id="edao" class="com.example.EmpDao">
674         <constructor-arg>
675           <ref bean="jtemplate"/>
676           </constructor-arg>
677       </bean>
678
```

```
679     </beans>
680
681   10)Test.java
682     package com.example;
683
684     import org.springframework.beans.factory.BeanFactory;
685     import org.springframework.beans.factory.xml.XmlBeanFactory;
686     import org.springframework.core.io.ClassPathResource;
687     import org.springframework.core.io.Resource;
688
689     public class SimpleTest {
690       public static void main(String[] args) {
691
692         Resource r=new ClassPathResource("applicationContext.xml");
693         BeanFactory factory=new XmlBeanFactory(r);
694
695         EmpDao dao=(EmpDao)factory.getBean("edao");
696         int status=dao.update(new Emp(23,"Tarun",35000));
697         System.out.println(status);
698       }
699     }
```