Spring Material: Part-4

Spring JDBC

How many types of application are there in JAVA?

➔By looking at the applications and the way we build the applications & by looking at the nature of the application that we are build these applications are classified into in three different groups.

1) Standalone Application or Desktop based applications.

➔The application is running in which it is installed (same System). It consumes the memory of the same systems & only one user will access that application.

2) Distributed or Thin browser based applications.

➔This application is again classified into multiple types.

i) Intra-Network Applications.

➔It will used by the employee of a particular organization. Outer world people will not access it.

ii) Internet Applications.

3) Batch processing systems.

Where should we use JDBC & where should we use Hibernate?

➔While working with JDBC if we want to process huge amount of record then we need to deal with a single JDBC ResultSet object.

➔While working with hibernate if you want to process huge amount of record then we need to take multiple objects representing multiple records.

➔This is a Problem and it may crash the system, while creating huge amount of objects.

➔Use JDBC for offline application that deals with huge amount of data where we don't need database portability feature.

➔While developing internet applications that deals with limited amount of personalized data then use Hibernate.

➔While developing internet application that performs batch processing of huge amount of data then use java JDBC.

Ex: Mobile number activation that is activates set of numbers at a time.

Spring JDBC

➔Spring JDBC provided abstraction layer on java JDBC and simplifies persistency logic development.

➔In Bigger application instead of using java JDBC it is recommended to use Spring JDBC.

What is the Difference between Spring JDBC & Java JDBC.

1st Difference: (Boiler Plate Logic)

➔Java JDBC is an API it does not provide boiler plate logic.

➔It increases the cost of development and increases the time requirement to develop an application. It also increases the chances of bug within your application.

➔When we use java JDBC only to retrieve the data from the data base we need to write minimum seven lines of code for executing & after it iterating, collecting, and populating the

data is going to take more amount of data. I.e. minimum 10 lines of code we need to write which may cause for increase the bugs.

➔Instead of going for java JDBC if we use spring JDBC maximum 2 lines of code we need to write to retrieve the data from the data base.

➔That means the whole boiler plate logic is taken care by spring JDBC.

➔Spring JDBC developer has provided classes which contains methods, which you call the boiler plate logic is being part of those classes and that executed and we use the require set of data. Maximum 1 line of code we need to write to retrieve huge amount of data from the data base such a way optimization will happen when you go for Spring JDBC. The boiler plate logic is taken care by the Spring JDBC. It is the first advantage for going for using spring JDBC as compare to the java JDBC.

2nd Difference: (Managing the Resources)

➔While working on Java JDBC programmer has to write the logic for load the class, create the connection, create the statement then execute the query in ResultSet.

➔If we opened several resources while executing then we have to write the logic for release/ close the resources.

➔If the resources are not being closed (connection or statement or ResultSet) then there is a chance of resource leakage problem will come.

➔But when it comes to Spring JDBC we don't require to open the connection, we don't need to create the statement we don't need to get the ResultSet.

➔SO if we don't create or open any resources we don't need to required close or manage those resources.

➔Everything will manage by Spring JDBC itself. Now your application will become out of zero or 100% free from resources leakage problem when you go for spring JDBC.

3rd Difference :(annoying try & catch block)

➔To make our java JDBC application work we must & should our code surrounded inside try and catch block.

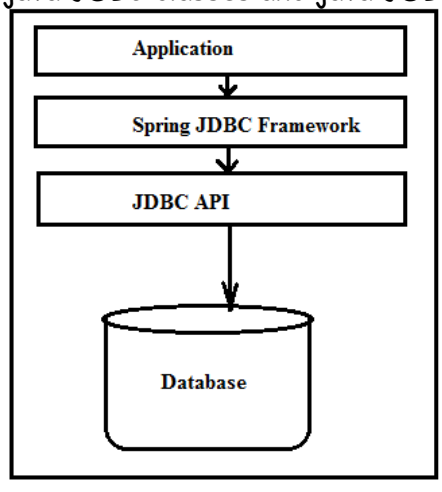
➔Because the java JDBC is always throws checked exception. So unless until we put them in to catch block your code will not get compile.

➔If there is no connection after catching if we don't have any alternate path of execution. Checked exceptions are always design to have alternate path of execution.

➔After writing the try and catch block and catching the exception also we don't have any alternate path of execution still we have need to write the try and catch block. Instead of it go for spring JDBC.

➔So all the exception that are in the spring JDBC is designed as unchecked exception.

➔So while you are working with spring JDBC your application classes will not directly talks with java JDBC classes rather our classes will talks with spring JDBC classes and spring JDBC classes are talks with java JDBC classes and java JDBC classes is talks with database.



➔While java JDBC talks to the database if any exception came then a SQL Exception has been reported by Database. Then database throws Exception to caller means java JDBC and throws it as SQL exception which is checked exception. Then Java JDBC throws back to the caller means spring JDBC class, So Spring JDBC has to catch the Exception.

➔ Then the spring JDBC is catches the exceptions and wraps the exceptions into spring JDBC Exceptions which are unchecked exception, SO we don't need to bother about to catch those exception whether we have alternate path or not.

➔Now we are free from writing the try and catch block. Which is benefits going for spring JDBC rather go for java JDBC.

➔All the Exception that are in the java JDBC are checked Exception .That is the problem in the java JDBC. The entire code is written in the java JDBC is inside the try & catch block which is annoying try and catch block.

➔Even if you don't have anything to do after catching you don't have any alternate path of execution still you have to catches the exception.
➔So it's a problem with java JDBC. Instead of it, if you go for spring JDBC all the Exception that are present there inside the spring JDBC are unchecked exceptions. So if you really want to handle the exception then only you need to write the try and catch block otherwise you will completely free from writing the try and catch block.

Spring Exceptions

➔Spring JDBC comes with bunch of Exception classes.
➔To classify the problem in your application and to easily know the cause of the problem Spring JDBC has provided bunch of exception classes which are called spring JDBC Exception classes.
➔For all spring JDBC Exception classes there is a parent classes called **DataAccsesExceptions**. Any Exception that are there in the spring JDBC will extend from DataAccsesExceptions.

4th Difference: (Transaction)

➔In java JDBC programmer has to write the logic for transaction. Instead of it if you go for spring JDBC the transaction management logic will be taken care by the spring JDBC itself.

Why should I go for Spring JDBC Why not Hibernate?

➔We never want to migrate our application from one database to another database. SO we don't want to use the hibernate for sacrificing the performance.

Why you don't want to migrate your application?

➔Suppose my application is use one database & the other applications are also using the same database.
➔There are 3 to 4 systems are built on the database, the same database is being centrally used by other party of the application also.
➔Suddenly migration is not a small thing. It is a huge migration.
➔Not only our application is to be migrate that is not enough when we migrate the database, all the other systems that are using the same database has to be migrated.
➔It is not possible to consider for rewriting the whole application that are there in register for sake of change in the database.

➔I hope that will not possible that is the reasons we are not using hibernate because we don't need to migrate the database so we use Spring JDBC as part of our application.

How to work with Spring JDBC?

➔If I wanted to work with accessing the data from the database using Spring JDBC.
➔Spring JDBC has provided 5 approaches in working with Spring JDBC to access the data from the database.

- ➔JDBC Template
- ➔Name parameters JDBC Template
- ➔Simple JDBC template
- ➔Simple JDBC insert or Simple JDBC call
- ➔Mapping SQL operations as sub classes

What are the possible operations in JDBC?

- ➔Data Definition Language (DDL)
- ➔Data Manipulation Language (DML) (insert,update,delete)
- ➔Data Query Language (DQL): (select query)
- ➔Data Transaction Language (DTL): (Commit, rollback)

Why in development environment people will not use DDL Operations?

➔Development starts with Database designing. Normalization, normal form, identify the entities (tables), deriving the relationship between those tables, normalize those tables into normal forms, and then ready to the data model with ER diagram.

➔During the start of the application 80 to 90% of the database design has to be completed. And anything that has been still left will be related to the unwanted requirements only.

➔Some requirements are not real and while during the progress of the application developments if those requirements are going to be clarifying then there will be a change in the database design, otherwise during the developments the database tables will not changed. Once the database model is finalized or completed then only developments of the application will happened.

➔Once the database model has been completed the design tables that we have come of with will be converted into SQL scripts. And these SQL script will be written in a file and it will be distributed with the whole team. The developments of the application will be happens using those tables only, the tables will not be created by the applications.

Why db Privileges not there with development?

➔To connect our application to the database we need url, user name, password, driver class will be provided by database administrator and will creates the schema.
➔In a development environment free access to the database will be available. Because we setup our own database instance here.
➔But when it goes to the test or a stage environment there will be a administrator who takes care of creating the instance of database to us. They will not provide the user name and password to our application where full privilege to access to the database .
➔So to create the username and password and give to us. Using this username and password we need to connect. Now if we want to access the database from the java application.

➔That means we need username and password of the data base which is provided by database administrator. We Creates the username and password with less privileges. Create privilege, drop privilege, and alter privilege these privileges will not be there for your user.
➔It means if you execute create table, table name with this username and password it will throws an Exception says this privilege is not available. Because java application is using this username and password. Customers are using those applications. If someone hacks your application data will be gone.
➔So the username and the password that has been allotted to your application to work with the database will not have any of the privileges. Zero privileges will be there apart from select, insert, update. Delete privilege is also not be there. All those privileges will be remote and the username and password will be given to you. With that only you code your application in java . In such a case how would you able to create the table within the database using it ? That is the worst thing if you do this using java.

Q. How to modify db table?

➔The interviewer will going to ask you did you ever got a chance to add a column into the database table did you ever tried to doing such a kind of things? If at all you get a chance based on a requirement if you may have to add a column into the database table what do you do or how do you add a column into the existing table?

➔We should add the column by writing alter scripts sending this query to database administrator. Ask the administrator to add the column. But it is wrong.what do you mean by it ?

➔So we have already an application has been developed which is running in the production environment or has been given to testing environment any of cases you can consider. Now my application has been developed in development environment. And the application has been tested in a QA environment. My development environment has database & my test environment also will have database. Now when I moved from development to QA I need to have tables being created in QA environment otherwise the code which we have written will not work without the tables.

➔Assume in my application one requirement is added or some enhancement is done by adding a extra column to the customer table by writing the SQL script. Assume we added an extra column name with alternate mobile no. Now the current code that I have here is for example 1.1 version in the development environment.

➔Now what do I need to do if I want to add a extra column in to the database table ? Its simple execute an alter query here in the database table then that code that is there in test environment will not consider this table . I write the logic for the persist the column into the data base table. Now the version my code is 1.2 . Now finally adding the column and modifying the code in my application , my application logic is now given to QA team for testing . But my code will not work. Because the code we have deployed here is having extra additional column which is not there in the QA database . So I should not do it. There is a process will be there .

➔The changes that I have making in adding the column , the relivant change I have made in java should push into all the environments of the same state. That coordination has to be doen. Otherwise the whole application is broken in all the environment.

➔That's why first alter query has to be written and will execute in the local environment and create an SQL script file with that alter query . Now test yourself that your code is working or not.
➔Now coming to your code into repository and then take the version no (1.2) now with this you send a email to database administrator, project manager and the lead saying this is an additional column that I have added into this corresponding table as per the enhancement requirement . And for this additional column that I have added into the table the corresponding code changes are been pushed into the svm repository and the version no of the code changes is 1.2 .
➔If you are deploying the code with version 1.1 then don't take this alter query but if you are taking the higher side of the code 1.2 and deploying it into the QA environment along with that execute the alter query , now the database will have the column here then only deploy the code .
➔This we have to email to the database administrator, lead, and project manager and to the deplorer. So the developer has to document this whole process to all the environment.

What is JDBC template ?

➔Spring JDBC is being classified into 4 packages .

- ➔Core package
- ➔Objects package
- ➔Util package
- ➔Exception package

When it comes to JDBC template there are two ways of working with JDBC template.

Classic approach

➔It is the old style which we come across. It is similar to java JDBC.
For example:In java JDBC programmer has to do everything like create the connection, create the statement, execute the query , get the result set . SO it is resembles like java JDBC approach when we work with classic approach. Even though API changes but the mechanism or working is same.

Query based approach

➔Now how we will working with java JDBC approach it is not like that. It is a modern way of working with Spring JDBC. We don't need to do anything like java JDBC. We just need to write the query and gives to the Spring JDBC template and it will give the data to you.

➔While working with any approach of Spring JDBC we need to use data source (Connection pooling) to interact with database software.

Using JDBC Template

➔This is central class of spring spring JDBC provides abstraction layer on java JDBC and simplifies persistency logic development.
➔This class takes care of common workflow activities of the JDBC Programming like opening connection, creating statement, closing the connection etc…
➔This allows programmer to avoid common error in application development like forgetting to close connection, statement.
➔To create JDBC template class object we need JDBC Data Source object. We need generally configure JDBC Template in spring bean configuration file and inject that object to DAO class.

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

      <!--
      <bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
            <property name="driverClassName" value="oracle.jdbc.driver.OracleDriver"/>
            <property name="url" value="jdbd:oracle:thin:@localhost:1521:xe"/>
            <property name="username" value="spr_usr"/>
            <property name="password" value="welcome1"/>
      </bean> -->

      <!-- Spring provided implementation class of BeanFactoryPostProcessor -->
      <bean id="bfpp" class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
            <property name="location" value="classpath:db.properties"/>
      </bean>

      <!-- Spring provided implementation class of DataSource -->
      <bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
            <property name="driverClassName" value="${db.driverClassName}"/>
            <property name="url" value="${db.url}"/>
            <property name="username" value="${db.username}"/>
            <property name="password" value="${db.password}"/>
      </bean>

      <!-- Create JDBC Template and pass input as dataSource -->
      <bean id="jdbcTemplate" class="org.springframework.jdbc.core.JdbcTemplate">
            <property name="dataSource" ref="dataSource"/>
      </bean>

      <!-- <bean id="projectDAO" class="com.sjf.dao.ProjectDAO" c:jdbcTemplate-ref="jdbcTemplate"/>
       -->
       <bean id="projectDAO" class="com.sjf.dao.ProjectDAO">
            <constructor-arg ref="jdbcTemplate"/>
       </bean>

</beans>
```

Callback Interface

➔The Method that is called by underlying jvm/container/server etc… automatically and dynamically is callback method.

➔The interface that contains such method is called callback interface.

➔Spring provides lots of callback interfaces along with lots of template classes.

➔We can use these callback interfaces to write customized logics to process data and to get data in customized manner.

➔Spring provides various callback interfaces of spring JDBC are…

StatementCallback ---→Allow to work with statement Object
PrepareStatementCallback
CallableStatementCallback
PrepareStatementCreator---→provide con object to create PrepareStatement
RowMapper---→To process 1 record of ResultSet at a time
ResultSetExtractor ---→To Process Multiple record of ResultSet at a time.

Note:

The implementation class methods of callback interface will be called by JdbcTemplate internally.

What is Call back mechanism ?

➔Let us assume technically this methodology. I have some piece of code & this code knows how to query the data from the database. But to query the data from the database this piece of code need connection. I can create a connection. But if am creating the connection I need to manage the connection. So I don't want to create the connection rather JDBC template can create the connection.

➔Now the require set of inputs that are required to executing my code is not there with me. It is with there JdbcTemplate . SO either I have to go to the jdbc template and take the connection. Now the connection is with me. But if I get the connection I need to manage the connection & the resources. So I don't need to get the connection.

➔ So now I have to give this code to the JdbcTemplate so that jdbc template takes care of carrying this code by passing the connection as input to carrying the operation. SO until the code completes the execution I need the connection.

➔JdbcTemplate is calling my code. So JdbcTemplate will know whether my code is completes its execution or not. So after execution of the code JdbcTemplate will close the connection automatically & we don't need to manage the resources.

➔So am not calling my code or methods to execute the operation. Jdbc template is calling my code by passing the required set of inputs as parameter to our methods. So if jdbc template has to call my method on my class I need to give the object of my class to jdbc template .

➔So I have to create the object of my class and give the object to JdbcTemplate by saying I have a method can you please call the method by passing the required set of input that's why my code will get executed. As am not calling my method and my code is handover to someone to calling the method that's the reason it is call CALLBACK MECHANISM.

Classic Approach Example-1(Dynamic Query)

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

	<!--
	<bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
		<property name="driverClassName" value="oracle.jdbc.driver.OracleDriver"/>
		<property name="url" value="jdbd:oracle:thin:@localhost:1521:xe"/>
		<property name="username" value="spr_usr"/>
		<property name="password" value="welcome1"/>
	</bean> -->

	<!-- Spring provided implementation class of BeanFactoryPostProcessor -->
	<bean id="bfpp" class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
		<property name="location" value="classpath:db.properties"/>
	</bean>

	<!-- Spring provided implementation class of DataSource -->
	<bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
		<property name="driverClassName" value="${db.driverClassName}"/>
		<property name="url" value="${db.url}"/>
		<property name="username" value="${db.username}"/>
		<property name="password" value="${db.password}"/>
	</bean>

	<!-- Create JDBC Template and pass input as dataSource -->
	<bean id="jdbcTemplate" class="org.springframework.jdbc.core.JdbcTemplate">
		<property name="dataSource" ref="dataSource"/>
	</bean>

	<!-- <bean id="projectDAO" class="com.sjf.dao.ProjectDAO" c:jdbcTemplate-ref="jdbcTemplate"/>
	 -->
	 <bean id="projectDAO" class="com.sjf.dao.ProjectDAO">
		<constructor-arg ref="jdbcTemplate"/>
	 </bean>

</beans>
```

```
public class ProjectDAO {
	private JdbcTemplate jdbcTemplate;
	public ProjectDAO(JdbcTemplate jdbcTemplate) {
		this.jdbcTemplate = jdbcTemplate;
	}

	public void getProject(final int projectNo) {
		PreparedStatementCreator creater= new PreparedStatementCreator() {
			private final String SQL_QUERY = "select * from project where project_No=?";
			@Override
			public PreparedStatement createPreparedStatement(Connection connection)
					throws SQLException {
				PreparedStatement pstmt = connection.prepareStatement(SQL_QUERY);
				pstmt.setInt(1,projectNo);
				return pstmt;
			}
		};

		PreparedStatementCallback< List<ProjectBo>> callBack=new PreparedStatementCallback<List<ProjectBo>>() {
			@Override
			public List<ProjectBo> doInPreparedStatement(PreparedStatement pstmt)
					throws SQLException, DataAccessException {
				List<ProjectBo> projectBoList = new ArrayList<ProjectBo>();
				ProjectBo projectBo = null;
				ResultSet rs = pstmt.executeQuery();
				while (rs.next()) {
					projectBo = new ProjectBo();
					projectBo.setProjectNo(rs.getInt("PROJECT_NO"));
					projectBo.setTitle(rs.getString("TITLE"));
					projectBo.setDescription(rs.getString("DESCR"));
					projectBo.setDuaration(rs.getString("DURATION"));
					projectBo.setStatus(rs.getString("STATUS"));
					projectBoList.add(projectBo);
				}
				return projectBoList;
			}
		};

		List<ProjectBo> projects=jdbcTemplate.execute(creater,callBack);
		for (ProjectBo projectBo : projects) {
			System.out.println(projectBo);
		}

	}
}
```

```
public class Test {
	public static void main(String[] args) throws SQLException {

		ApplicationContext context=new ClassPathXmlApplicationContext("com/sjf/common/application-context.xml");
		ProjectDAO projectDAO=context.getBean("projectDAO",ProjectDAO.class);
		projectDAO.getProject(101);
	}
}
```

Internals

➔Most of time our logic depends upon callback mechanism. Whatever we wanted to do in classic approach will be divided in to 2 phases of execution.

➔First phase of execution is called preparing phase (Creator phase). That what you need to execute your code write all this code under preparing phase

➔Second phase of execution is called callback phase(Callback phase). The execution of the code will happens in callback phase. Or collecting the results.

➔Every piece of code that you are writing under Spring Jdbc classic approach will falls under 2 phase

➔Like java JDBC in spring JDBC there are 2 statements to execute the query.

1) Statement
2) Prepared Statement

➔Now there are 2 phases where our code is divided.

i) Creator phase
ii) Callback phase

➔In creator phase we need to create the prepared statement and in callback phase we need to execute the prepared statement and get the results.

➔Now to create the statement we need connection. But we don't want to create the connection because if we create the connection we need to manage it.

➔ If I don't create the connection then jdbc template has to provide the connection.How jdbc template provide the connection ?

➔To get the connection from the jdbc template we need to write a class and inside the class we need to write a method by passing connection as parameter to that method.

➔Jdbc template has to call that method . But how does the jdbc template know your class has that method to call by passing connection as parameter?

➔That's why every class that we wanted to give to the jdbc template to get it executed should implements from standard interfaces that are provided by jdbc template . So we need to write 2 pices of logic. For creator phase and callback phase.

➔In creator phase we need to create preparedStatement & in callback phase we need to execute the prepared statement. That's why spring jdbc has provided 2 class to us which are interfaces.

➔PreparedStatementCreator(I)

➔And the method inside of this interface is :

```
public PreparedStatement createPreparedStatement(Connection connection)
```

➔PreparedStatement Callback(I)

➔And the method inside of the interface is :

```
public ProjectBo doInPreparedStatement(PreparedStatement pstmt
```

➔In creator phase we use the connection, creates the preparedstatement and give to jdbc template . Jdbc template is caaling this method , after calling this method jdbc template will passes the connection and calls our method .

➔Then our code will creates the preparedstatement and give to jdbc template . Then jdbc template will passes that to 2 implements statement by passing preparedstatement asking him to using. Then you will execute it and return the value to jdbc template .

➔Now jdbc template will understand you completed using the connection , you completed using the statement and after then both will closed by the jdbc template .

➔Now managing the connection and resources is taken care by jdbc template.

Difference between Statement & PreparedStatement

➔**The main reason why people go for preparedStatement is to avoid SQL Injection**. SO when are you going to append the values into the sql queries means when the query need inputs . SO when there is a dynamic sql query don't use statement use preparedstatement .

➔So at the time when we are going to create prepare statement , con.preparestatement(SQL Query) and when we set parameters while we call execute() method , it checks is the data we have submitted is itself a query or not , the validation will be done by preparedstatement .

➔If at all the input that we have substituted for the? Is an another query it will throws an exception avoiding you to execute the query .

➔If my query is a static query (no where clause is there) and I want to execute the query only once within the one session then use of statement and prepared statement is same there is no difference.

➔I have a static sql query which needs repeatedly get executed with in the same session or I have dynamic query which may have executed either once or multiple times in this go for preparedstatement only don't use statement .

Statement creation in spring jdbc:

➔If there is a static query it means we don't have any dynamic inputs to be substituted and you want to execute this query only once then instead of going for preparedstatement we can use statement.

Classic Approach Example-2(Dynamic Query)

```
public class ProjectBo {
	private int projectNo;
	private String title;
	private String description;
	private String duaration;
	private String status;
	//setters & gettters
}
```

db.properties

```
db.driverClassName=oracle.jdbc.OracleDriver
db.url=jdbc:oracle:thin:@localhost:1521:xe
db.username=spr_usr
db.password=welcome1
```

```
public class Test {
	public static void main(String[] args) throws SQLException {

		ApplicationContext context=
			new ClassPathXmlApplicationContext("com/sjf/common/application-context.xml");
		ProjectDAO projectDAO=context.getBean("projectDAO",ProjectDAO.class);
		projectDAO.getAllProject();
	}
}
```

```
public class ProjectDAO {
	private JdbcTemplate jdbcTemplate;
	public ProjectDAO(JdbcTemplate jdbcTemplate) {
		this.jdbcTemplate = jdbcTemplate;
	}
	public void getAllProject() {
		List<ProjectBo> projectBoList = jdbcTemplate.execute(new ProjectStatementCallback());
		for (ProjectBo projectBo : projectBoList) {
			System.out.println(projectBo);
		}
	}
	private final class ProjectStatementCallback implements
			StatementCallback<List<ProjectBo>> {
		private final String SQL_QUERY = "select * from project";

		@Override
		public List<ProjectBo> doInStatement(Statement pstmt)throws SQLException, DataAccessException {
			List<ProjectBo> projectBoList = new ArrayList<ProjectBo>();
			ProjectBo projectBo = null;
			ResultSet rs = pstmt.executeQuery(SQL_QUERY);
			while (rs.next()) {
				projectBo = new ProjectBo();
				projectBo.setProjectNo(rs.getInt("PROJECT_NO"));
				projectBo.setTitle(rs.getString("TITLE"));
				projectBo.setDescription(rs.getString("DESCR"));
				projectBo.setDuaration(rs.getString("DURATION"));
				projectBo.setStatus(rs.getString("STATUS"));
				projectBoList.add(projectBo);
			}
			return projectBoList;
		}

	}
}
```

Static Query

Query Base Approach

➔There are several methods are there in JDBC Template.

Select Operations

- **queryForInt()**

  select count(*) from project

  ➔This is used to execute query that gives single numeric values as result.

  ➔There are three over loaded Methods.

```
public int queryForInt(String sql)
public int queryForInt(String sql, Object[] args)
public int queryForInt(String sql, Object[] args, int[] argTypes)
```

- **queryForMap()**

  select * from project where project_no=1

  ➔This is used to execute query that gives single record as result.

  ➔There are three over loaded Methods.

```
public Map<String, Object> queryForMap(String sql)
public Map<String, Object> queryForMap(String sql, Object[] args)
public Map<String, Object> queryForMap(String sql, Object[] args,int[] argTypes)
```

,

- **queryForObject()**

  select * from project where project_no=1

  ➔This is used to execute query and to get result into Bo class object.

```
public <T> T queryForObject(String sql, RowMapper<T> rowMapper)
public <T> T queryForObject(String sql, Object[] args,RowMapper<T> rowMapper)
```

- **queryForList()**

  select project_No,Title from project where project_no=1

  ➔This is used to execute query that gives multiple records.

```
public List<Map<String, Object>> queryForList(String sql)
```

Non-Select Operations

- **update()**
- **batchUpdate()**

Query Base Approach Example-1

```
public class Test {
	public static void main(String[] args) throws SQLException {

		ApplicationContext context=
				new ClassPathXmlApplicationContext("com/sjf/common/application-context.xml");
		ProjectDAO projectDAO=context.getBean("projectDAO",ProjectDAO.class);
		projectDAO.getNoOfProject();
	}
}
```

```
public class ProjectDAO {
	private JdbcTemplate jdbcTemplate;
	private final String SQL_NO_OF_PROJECT="SELECT COUNT(*) FROM PROJECT";
	public ProjectDAO(JdbcTemplate jdbcTemplate) {
		this.jdbcTemplate = jdbcTemplate;
	}

	public void getNoOfProject() {
		int count=jdbcTemplate.queryForInt(SQL_NO_OF_PROJECT);
		System.out.println("Total No of Project : "+count);
	}
}
```

Query Base Approach Example-2(QueryForObject() for primitives)

```
public class Test {
	public static void main(String[] args) throws SQLException {

		ApplicationContext context=
				new ClassPathXmlApplicationContext("com/sjf/common/application-context.xml");
		ProjectDAO projectDAO=context.getBean("projectDAO",ProjectDAO.class);
		projectDAO.getNoOfProject(101);
	}
}
```

```
public class ProjectDAO {
	private JdbcTemplate jdbcTemplate;
	private final String SQL_NO_OF_PROJECT="SELECT count(*) FROM PROJECT WHERE PROJECT_NO=?";
	public ProjectDAO(JdbcTemplate jdbcTemplate) {
		this.jdbcTemplate = jdbcTemplate;
	}

	public void getNoOfProject(int projectNo) {
		int count=jdbcTemplate.queryForObject(SQL_NO_OF_PROJECT , Integer.class , new Object[]{projectNo});
		System.out.println("Total No of Project : "+count);
	}
}
```

Query Base Approach Example-3(QueryForObject() to get Our own classes)

```
public class Test {
	public static void main(String[] args) throws SQLException {

		ApplicationContext context=
				new ClassPathXmlApplicationContext("com/sjf/common/application-context.xml");
		ProjectDAO projectDAO=context.getBean("projectDAO",ProjectDAO.class);
		projectDAO.getNoOfProject(101);
	}
}
```

```
public class ProjectDAO {
	private JdbcTemplate jdbcTemplate;
	private final String SQL_NO_OF_PROJECT="SELECT count(*) FROM PROJECT WHERE PROJECT_NO=?";
	public ProjectDAO(JdbcTemplate jdbcTemplate) {
		this.jdbcTemplate = jdbcTemplate;
	}

	public void getNoOfProject(int projectNo) {
		int count=jdbcTemplate.queryForObject(SQL_NO_OF_PROJECT , Integer.class , new Object[]{projectNo});
		System.out.println("Total No of Project : "+count);
	}
}
```

```
public class ProjectDAO {
	private JdbcTemplate jdbcTemplate;
	private final String SQL_NO_OF_PROJECT="SELECT * FROM PROJECT WHERE PROJECT_NO=?";
	public ProjectDAO(JdbcTemplate jdbcTemplate) {
		this.jdbcTemplate = jdbcTemplate;
	}

	public void findProject(int projectNo) {
		ProjectBo bo=jdbcTemplate.queryForObject(SQL_NO_OF_PROJECT , new ProjectRowMapper() , new Object[]{projectNo});
		System.out.println(bo);
	}
	private final class ProjectRowMapper implements RowMapper<ProjectBo>{
		@Override
		public ProjectBo mapRow(ResultSet rs, int row) throws SQLException {
			ProjectBo bo=new ProjectBo();
			bo.setProjectNo(rs.getInt(1));
			bo.setTitle(rs.getString(2));
			bo.setDescription(rs.getString(3));
			bo.setDuaration(rs.getInt(4));
			bo.setStatus(rs.getString(5));
			return bo;
		}
	}
}
```

Query Base Approach Example-4(Query for Get all the Project Records)

```
public class Test {
	public static void main(String[] args) throws SQLException {

		ApplicationContext context=
				new ClassPathXmlApplicationContext("com/sjf/common/application-context.xml");
		ProjectDAO projectDAO=context.getBean("projectDAO",ProjectDAO.class);
		projectDAO.getAllProjects();
	}
}
```

```
public class ProjectDAO {
	private JdbcTemplate jdbcTemplate;
	private final String SQL_NO_OF_PROJECT="SELECT * FROM PROJECT";
	public ProjectDAO(JdbcTemplate jdbcTemplate) {
		this.jdbcTemplate = jdbcTemplate;
	}

	public void getAllProjects() {
		List<ProjectBo> boList=jdbcTemplate.queryForObject(SQL_NO_OF_PROJECT , new ProjectRowMapper());
		for (ProjectBo projectBo : boList) {
			System.out.println(projectBo);
		}
	}
	private final class ProjectRowMapper implements RowMapper<List<ProjectBo>>{
		@Override
		public List<ProjectBo> mapRow(ResultSet rs, int row) throws SQLException {
			List<ProjectBo> boList=new ArrayList<ProjectBo>();
			while (rs.next()) {
				ProjectBo bo=new ProjectBo();
				bo.setProjectNo(rs.getInt(1));
				bo.setTitle(rs.getString(2));
				bo.setDescription(rs.getString(3));
				bo.setDuaration(rs.getInt(4));
				bo.setStatus(rs.getString(5));
				boList.add(bo);
			}
			return boList;
		}
	}
}
```

Query Base Approach Example-5(Select Query for specific column of a record)

```
public class Test {
	public static void main(String[] args) throws SQLException {

		ApplicationContext context=
				new ClassPathXmlApplicationContext("com/sjf/common/application-context.xml");
		ProjectDAO projectDAO=context.getBean("projectDAO",ProjectDAO.class);
		projectDAO.getProjectDuration();
	}
}
```

```
public class ProjectDAO {
	private JdbcTemplate jdbcTemplate;
	private final String SQL_FOR_PROJECT="SELECT TITLE,DURATION FROM PROJECT";
	public ProjectDAO(JdbcTemplate jdbcTemplate) {
		this.jdbcTemplate = jdbcTemplate;
	}

	public void getProjectDuration() {
		List<Map<String, Object>> boList=jdbcTemplate.queryForList(SQL_FOR_PROJECT);
		for (Map<String, Object> bo : boList) {
			for (String key :bo.keySet()) {
				System.out.print(key+"="+bo.get(key)+"\t");
			}
			System.out.println();
		}
	}
```

Query Base Approach Example-6

(Get all the Project Records by using query()

```
public class ProjectDAO {
	private JdbcTemplate jdbcTemplate;
	private final String SQL_FOR_PROJECT="SELECT * FROM PROJECT";
	public ProjectDAO(JdbcTemplate jdbcTemplate) {
		this.jdbcTemplate = jdbcTemplate;
	}

	public void getProjects() {
		List<List<ProjectBo>> boList=jdbcTemplate.query(SQL_FOR_PROJECT,new ProjectRowMapper());
		for (List<ProjectBo> list : boList) {
			for (ProjectBo projectBo : list) {
				System.out.println(projectBo);
			}
			System.out.println();
		}

	}
	private final class ProjectRowMapper implements RowMapper<List<ProjectBo>>{
		@Override
		public List<ProjectBo> mapRow(ResultSet rs, int row) throws SQLException {
			List<ProjectBo> boList=new ArrayList<ProjectBo>();
			while (rs.next()) {
				ProjectBo bo=new ProjectBo();
				bo.setProjectNo(rs.getInt(1));
				bo.setTitle(rs.getString(2));
				bo.setDescription(rs.getString(3));
				bo.setDuaration(rs.getInt(4));
				bo.setStatus(rs.getString(5));
				boList.add(bo);
			}
			return boList;
		}
	}
}
```

Query Base Approach Example-7(Pagination)

```
public class Test {
	public static void main(String[] args) throws SQLException {

		ApplicationContext context=
				new ClassPathXmlApplicationContext("com/sjf/common/application-context.xml");
		ProjectDAO projectDAO=context.getBean("projectDAO",ProjectDAO.class);
		projectDAO.getProjects(3,1);
	}
}
```

```
public class ProjectDAO {
	private JdbcTemplate jdbcTemplate;
	private final String SQL_FOR_PROJECT="SELECT * FROM PROJECT ORDER BY TITLE";
	public ProjectDAO(JdbcTemplate jdbcTemplate) {
		this.jdbcTemplate = jdbcTemplate;
	}

	public void getProjects(int pageSize,int pageNo ) {
		List<ProjectBo> boList=jdbcTemplate.query(SQL_FOR_PROJECT,new ProjectResultSetExtractor(pageSize, pageNo));
		for (ProjectBo bo : boList) {
			System.out.println(bo);
		}

	}
	private final class ProjectResultSetExtractor implements ResultSetExtractor<List<ProjectBo>>{
		private int pageSize;
		private int pageNo;

		public ProjectResultSetExtractor(int pageSize,int pageNo) {
			this.pageSize=pageSize;
			this.pageNo=pageNo;
		}

		@Override
		public List<ProjectBo> extractData(ResultSet rs) throws SQLException,
				DataAccessException {

			int startIndex=(pageSize *(pageNo-1))+1;
			int endIndex=pageSize*pageNo;
			int rowIndex=1;
			ProjectBo project=null;
			List<ProjectBo> projects=null;

			projects=new ArrayList<ProjectBo>();

			while (rs.next() && rowIndex<=endIndex) {
				if (rowIndex>=startIndex) {
					project=new ProjectBo();
					project.setProjectNo(rs.getInt(1));
					project.setTitle(rs.getString(2));
					project.setDescription(rs.getString(3));
					project.setDuaration(rs.getInt(4));
					project.setStatus(rs.getString(5));
					projects.add(project);
				}
				rowIndex++;
			}
			return projects;
		}
	}
}
```

Output

```
Project [projectNo=106, title=Airport Management System, description=Ams, duaration=60, status=Test]
Project [projectNo=101, title=BookService, description=Book Sale in Online, duaration=60, status=Development]
Project [projectNo=105, title=Hostel Management System, description=Hms, duaration=50, status=Test]
```

Query Base Approach Example-8

How to get Auto generated Primary key after insert

```
public class Test {
	public static void main(String[] args) throws SQLException {

		ApplicationContext context=
				new ClassPathXmlApplicationContext("com/sjf/common/application-context.xml");
		ProjectDAO projectDAO=context.getBean("projectDAO",ProjectDAO.class);
		projectDAO.InsertProjects();

	}
}
```

```
public class ProjectDAO {
	private JdbcTemplate jdbcTemplate;
	private final String SQL_FOR_PROJECT="INSERT INTO PROJECT VALUES (sampleseq.nextval,?,?,?,?)";
	public ProjectDAO(JdbcTemplate jdbcTemplate) {
		this.jdbcTemplate = jdbcTemplate;
	}

	public void InsertProjects() {
		GeneratedKeyHolder keys=new GeneratedKeyHolder();

		ProjectBo bo=new ProjectBo();
		bo.setTitle("Hospital Management System");
		bo.setDescription("HMS");
		bo.setDuaration(45);
		bo.setStatus("Test");

		Object key=jdbcTemplate.update(new ProjectPrepareStatementCreater(bo), keys);
		System.out.println("Inserted Primary Key is:"+keys.getKey());
	}
	private final class ProjectPrepareStatementCreater implements PreparedStatementCreator{
		private ProjectBo bo=null;
		public ProjectPrepareStatementCreater(ProjectBo bo) {
			this.bo=bo;
		}
		@Override
		public PreparedStatement createPreparedStatement(Connection con)throws SQLException {
			PreparedStatement pstmt=con.prepareStatement(SQL_FOR_PROJECT, new String[]{"project_no"});
			pstmt.setString(1, bo.getTitle());
			pstmt.setString(2, bo.getDescription());
			pstmt.setInt(3, bo.getDuaration());
			pstmt.setString(4, bo.getStatus());
			return pstmt;
		}
	}
}
```

NamedParameterJdbcTemplet

➔JdbcTemplate Supports only positional parameters(?).
➔We need object[]{} values to assign values to positional parameters based on there indexes
➔In this process if we confuse towards identifying the indexes there is a possibility of getting the problem.
➔To overcome this problem the spring JDBc has given NamedParameterJdbcTemplate as wrappee around JdbcTemplate supporting NamedParameter on the query.
➔This provides all the functionalities of JdbcTemplate but allows us to work with named parameter(:<variableName>)

There are three ways to set values to NamedParameter

➔As Map<String,Object> obj=new Hashmap<String,Object>();
Obj.put(key,value);

➔By using implementation class object of SqlParameterSource(I)
MapSqlParameterSource allow to specify the name , values of named parameters as the args of add(-,-).

➔BeanPropertySqlParameterSource allows to specify bean object values the named parameter values and the names of named parameter and BeanProperty names must match.

➔To get Few Column

queryforList

```
public class Test {
	public static void main(String[] args) throws SQLException {
		ApplicationContext context=
				new ClassPathXmlApplicationContext("com/sjf/common/application-context.xml");
		ProjectDAO projectDAO=context.getBean("projectDAO",ProjectDAO.class);
		List<Map<String, Object>> boList=projectDAO.findProjects(101);
		for (Map<String, Object> map : boList) {
			for (String key: map.keySet()) {
				System.out.print(map.get(key)+"\t");
			}
			System.out.println();
		}
	}
}
```

```
public class ProjectDAO {
	private NamedParameterJdbcTemplate npjdbcTemplate;
	private final String SQL_FOR_PROJECT =
			"SELECT TITLE,DESCR,DURATION FROM PROJECT WHERE PROJECT_NO= :projectNo";

	public ProjectDAO(NamedParameterJdbcTemplate npjdbcTemplate) {
		this.npjdbcTemplate = npjdbcTemplate;
	}
	public List<Map<String, Object>> findProjects(int projectNo) {

		MapSqlParameterSource paramSource = new MapSqlParameterSource();
		paramSource.addValue("projectNo", projectNo);
		return npjdbcTemplate.queryForList(SQL_FOR_PROJECT, paramSource);
	}
}
```

➔To get complete Row of a Project

```
public class Test {
	public static void main(String[] args) throws SQLException {
		ApplicationContext context=
				new ClassPathXmlApplicationContext("com/sjf/common/application-context.xml");
		ProjectDAO projectDAO=context.getBean("projectDAO",ProjectDAO.class);
		List<ProjectBo> projects=projectDAO.findProjects(101);
		System.out.println(projects);
	}
}
```

```
public class ProjectDAO {
	private NamedParameterJdbcTemplate npjdbcTemplate;
	private final String SQL_FOR_PROJECT = "SELECT * FROM PROJECT WHERE PROJECT_NO= :projectNo";

	public ProjectDAO(NamedParameterJdbcTemplate npjdbcTemplate) {
		this.npjdbcTemplate = npjdbcTemplate;
	}

	public List<ProjectBo> findProjects(int projectNo) {

		Map<String, Object> paramSource = new HashMap<String, Object>();
		paramSource.put("projectNo", projectNo);
		return npjdbcTemplate.query(SQL_FOR_PROJECT, paramSource,
				new ProjectRowMapper());
	}

	private final class ProjectRowMapper implements
			org.springframework.jdbc.core.RowMapper<ProjectBo> {
		@Override
		public ProjectBo mapRow(ResultSet rs, int rowIndex) throws SQLException {
			ProjectBo project = new ProjectBo();
			project.setProjectNo(rs.getInt(1));
			project.setTitle(rs.getString(2));
			project.setDescription(rs.getString(3));
			project.setDuaration(rs.getInt(4));
			project.setStatus(rs.getString(5));
			return project;
		}
	}
}
```

```
?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd
		http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spring-util-3.0.xsd">
	<!-- Spring provided implementation class of BeanFactoryPostProcessor -->
	<bean id="bfpp" class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
		<property name="location" value="classpath:db.properties"/>
	</bean>
	<!-- Spring provided implementation class of DataSource -->
	<bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
		<property name="driverClassName" value="${db.driverClassName}"/>
		<property name="url" value="${db.url}"/>
		<property name="username" value="${db.username}"/>
		<property name="password" value="${db.password}"/>
	</bean>
	<!-- Create NamedParameterJDBC Template and pass input as dataSource -->
	<bean id="namedjdbcTemplate" class="org.springframework.jdbc.core.namedparam.NamedParameterJdbcTemplate">
		<constructor-arg ref="dataSource"/>
	</bean>
	 <bean id="projectDAO" class="com.sjf.dao.ProjectDAO">
		<constructor-arg ref="namedjdbcTemplate"/>
	 </bean>
</beans>
```

Output

```
[Project [projectNo=101, title=BookService, description=Book Sale in Online, duration=60, status=Development]]
```

➔To get one Column (queryforObject)

```
public class ProjectDAO {
	private NamedParameterJdbcTemplate npjdbcTemplate;
	private final String SQL_FOR_PROJECT = "SELECT TITLE FROM PROJECT WHERE PROJECT_NO= :projectNo";

	public ProjectDAO(NamedParameterJdbcTemplate npjdbcTemplate) {
		this.npjdbcTemplate = npjdbcTemplate;
	}
	public String findProjects(int projectNo) {

		MapSqlParameterSource paramSource = new MapSqlParameterSource();
		paramSource.addValue("projectNo", projectNo);
		return npjdbcTemplate.queryForObject(SQL_FOR_PROJECT, paramSource,
				String.class);
	}
}
```

➔SqlParameterSource

```
public class ProjectDAO {
	private NamedParameterJdbcTemplate npjdbcTemplate;
	private final String SQL_FOR_PROJECT = "SELECT * FROM PROJECT WHERE PROJECT_NO= :projectNo";

	public ProjectDAO(NamedParameterJdbcTemplate npjdbcTemplate) {
		this.npjdbcTemplate = npjdbcTemplate;
	}
	public List<ProjectBo> findProjects(int projectNo) {

		MapSqlParameterSource paramSource = new MapSqlParameterSource();
		paramSource.addValue("projectNo", projectNo);
		return npjdbcTemplate.query(SQL_FOR_PROJECT, paramSource,
				new ProjectRowMapper());
	}
	private final class ProjectRowMapper implements
			org.springframework.jdbc.core.RowMapper<ProjectBo> {
		@Override
		public ProjectBo mapRow(ResultSet rs, int rowIndex) throws SQLException {
			ProjectBo project = new ProjectBo();
			project.setProjectNo(rs.getInt(1));
			project.setTitle(rs.getString(2));
			project.setDescription(rs.getString(3));
			project.setDuaration(rs.getInt(4));
			project.setStatus(rs.getString(5));
			return project;
		}
	}
}
```

➔Insert using NamedSqlParameterSource

```
public class Test {
	public static void main(String[] args) throws SQLException {
		ApplicationContext context=
				new ClassPathXmlApplicationContext("com/sjf/common/application-context.xml");
		ProjectDAO projectDAO=context.getBean("projectDAO",ProjectDAO.class);
		ProjectBo bo=new ProjectBo();
		bo.setTitle("Hospital Management System");
		bo.setDescription("HMS");
		bo.setDuaration(45);
		bo.setStatus("Test");
		System.out.println(projectDAO.InsertProjects(bo));
	}
}
```

```
public class ProjectDAO {
	private NamedParameterJdbcTemplate jdbcTemplate;
	private final String SQL_FOR_PROJECT="INSERT INTO PROJECT
                                              VALUES (:projectNo,:title,:description,:duration,:status)";
	public ProjectDAO(NamedParameterJdbcTemplate jdbcTemplate) {
		this.jdbcTemplate = jdbcTemplate;
	}
	public int InsertProjects(ProjectBo bo) {
		MapSqlParameterSource paramSource=new MapSqlParameterSource();
		paramSource.addValue("projectNo", bo.getProjectNo());
		paramSource.addValue("title", bo.getTitle());
		paramSource.addValue("description", bo.getDescription());
		paramSource.addValue("duration",bo.getDuaration());
		paramSource.addValue("status",bo.getStatus());
		int rowCount=jdbcTemplate.update(SQL_FOR_PROJECT, paramSource);
		return rowCount;
	}
}
```

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

	<!-- Spring provided implementation class of BeanFactoryPostProcessor -->
	<bean id="bfpp" class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
		<property name="location" value="classpath:db.properties"/>
	</bean>

	<!-- Spring provided implementation class of DataSource -->
	<bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
		<property name="driverClassName" value="oracle.jdbc.driver.OracleDriver"/>
		<property name="url" value="jdbd:oracle:thin:@localhost:1521:xe"/>
		<property name="username" value="spr_usr"/>
		<property name="password" value="welcome1"/>
	</bean>

	<!-- Create NamedParameterJDBC Template and pass input as dataSource -->
	<bean id="jdbcTemplate" class="org.springframework.jdbc.core.namedparam.NamedParameterJdbcTemplate">
		<constructor-arg ref="dataSource"/>
	</bean>

	 <bean id="projectDAO" class="com.sjf.dao.ProjectDAO">
		<constructor-arg ref="jdbcTemplate"/>
	 </bean>
</beans>
```

BeanPropertySqlParameterSource

```
public class Test {
	public static void main(String[] args) throws SQLException {
		ApplicationContext context=
				new ClassPathXmlApplicationContext("com/sjf/common/application-context.xml");
		ProjectDAO projectDAO=context.getBean("projectDAO",ProjectDAO.class);
		ProjectBo bo=new ProjectBo();
		bo.setProjectNo(111);
		bo.setTitle("Hospital Management System");
		bo.setDescription("HMS");
		bo.setDuaration(45);
		bo.setStatus("Test");
		System.out.println(projectDAO.InsertProjects(bo));
	}
}
```

```
public class ProjectDAO {
	private NamedParameterJdbcTemplate jdbcTemplate;
	private final String SQL_FOR_PROJECT="INSERT INTO PROJECT VALUES (:projectNo,:title,:description,:duaration,:status)";
	public ProjectDAO(NamedParameterJdbcTemplate jdbcTemplate) {
		this.jdbcTemplate = jdbcTemplate;
	}
	public int InsertProjects(ProjectBo bo) {
		BeanPropertySqlParameterSource paramSource=new BeanPropertySqlParameterSource(bo);

		/*MapSqlParameterSource paramSource=new MapSqlParameterSource();
		paramSource.addValue("projectNo", bo.getProjectNo());
		paramSource.addValue("title", bo.getTitle());
		paramSource.addValue("description", bo.getDescription());
		paramSource.addValue("duration",bo.getDuaration());
		paramSource.addValue("status",bo.getStatus());*/

		int rowCount=jdbcTemplate.update(SQL_FOR_PROJECT, paramSource);
		return rowCount;
	}
}
```

SimpleJdbcInsert

➔It uses database table metadata concepts are used to insert record dynamically without insert SQL Query . It just take table name, column name, column values from the programmers and uses them to generate dynamic SQL> insert query to insert record.
➔We can give these column names and values either as Map object or as SQLParameterSource implementation class Object.
➔The Map Object contains column name as keys and column values as values.
➔To create SimpleJdbcInsert object DataSource object is dependant object.
➔This class makes as database Independent. We can switch one database to another database easily.

```
public class Test {
	public static void main(String[] args) throws SQLException {
		ApplicationContext context=
				new ClassPathXmlApplicationContext("com/sjf/common/application-context.xml");
		ProjectDAO projectDAO=context.getBean("projectDAO",ProjectDAO.class);
		ProjectBo bo=new ProjectBo();
		bo.setProjectNo(19);
		bo.setTitle("Hospital Management System");
		bo.setDescription("HMS");
		bo.setDuration(45);
		bo.setStatus("Test");
		System.out.println(projectDAO.InsertProjects(bo));
	}
}
```

```
public class ProjectDAO {
	private SimpleJdbcInsert jdbcTemplate;
	public ProjectDAO(SimpleJdbcInsert jdbcTemplate) {
		this.jdbcTemplate = jdbcTemplate;
	}
	public int InsertProjects(ProjectBo bo) {
		BeanPropertySqlParameterSource paramSource=new BeanPropertySqlParameterSource(bo);
		jdbcTemplate.setTableName("PROJECT");
		int rowCount=jdbcTemplate.execute(paramSource);
		return rowCount;
	}
}
```

Mapping SQL operations as sub classes

➔This is call technique in spring jdbc to perform persistence operation. This technique makes programmer to develope sub classes where we set tha datasource, declaring parameters and compile query only one time but we execute query multiple times by reusing the object of sub class for multiple times.

➔This concept is similar to JDO (Java Data Object) programming. Spring JDBC supplies SqlQuery,SqlUpdate classes to develop these sub classes.

➔Use SqlQuery for select operation and use SqlUpdate for non-select operations.

SqlQuery

➔We develop separator subclass extending from SqlQuery for select query inside DAO class.

➔This subclass supplies DataSource query to super class to execute query for multiple times by compiling the query only for one time.

➔SqlQuery supplies execute() method to execute query that gives the List of objects and also give finder() method to execute query that gives only one Object(One Record).

➔This class also gives maprow(--,--) method which can be used to process each record of ResultSet object in to another object.

➔The execute(),finder() method internally it calls mapRow() method.

➔Since SqlQuery is abstract class having more abstract methods we prefer using 'MappingSqlQuery' which is subclass in the inheritance hierarchy of SqlQuery.

➔While developing subclass for SqlQuery/MappingSqlQuery classes we use

- ➔declareParameter(-);
- ➔compile()

➔This technique say's give everything to super class from subclass for query execution and reuse subclass object.

MappingSqlQuery

```
public class Test {
        public static void main(String[] args) throws SQLException {
                ApplicationContext context=
                                new ClassPathXmlApplicationContext("com/sjf/common/application-context.xml");
                ProjectDAO projectDAO=context.getBean("projectDAO",ProjectDAO.class);
                List<ProjectBo> boList=projectDAO.findProjects(101);
                for (ProjectBo bo: boList) {
                        System.out.println(bo);
                }
        }
}
```

```java
public class ProjectDAO {
	private DataSource dataSource;
	private ProjectMappingSqlQuery pmsq;
	private final String SQL_FOR_PROJECT =
			"SELECT * FROM PROJECT WHERE PROJECT_NO= :projectNo";

	/**
	 *In side  Constructor we have to pass the data Source
	 * @param dataSource
	 */
	public ProjectDAO(DataSource dataSource) {
		this.dataSource=dataSource;
		pmsq=new ProjectMappingSqlQuery(dataSource, SQL_FOR_PROJECT);
	}

	/**
	 * This method to get the Project by projectNo.
	 * @param projectNo
	 * @return
	 */
	public List<ProjectBo> findProjects(int projectNo) {
		return pmsq.execute(new Object[]{projectNo});
	}



	/**
	 * ProjectRowMapper by which we need to pass the DataSource and SQL Query to super class
	 * @author Dhananjaya
	 *
	 */
	private final class ProjectMappingSqlQuery extends MappingSqlQuery<ProjectBo>{
		public ProjectMappingSqlQuery(DataSource dataSource,String sqlQuery){
			super(dataSource,sqlQuery);
			declareParameter(new SqlParameter(java.sql.Types.INTEGER));
			compile();
		}
		@Override
		protected ProjectBo mapRow(ResultSet rs, int roIndex)throws SQLException {
			ProjectBo projectBo=new ProjectBo();
			projectBo.setProjectNo(rs.getInt(1));
			projectBo.setTitle(rs.getString(2));
			projectBo.setDescription(rs.getString(3));
			projectBo.setDuaration(rs.getInt(4));
			projectBo.setStatus(rs.getString(5));
			return projectBo;
		}
	}
}
```

```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"
	xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd
		http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spring-util-3.0.xsd">

	<bean id="dataSource" 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="spr_usr"/>
		<property name="password" value="welcome1"/>
	</bean>
	<bean id="projectDAO" class="com.sjf.dao.ProjectDAO">
		<constructor-arg ref="dataSource"/>
	 </bean>
</beans>
```

OutPut

```
Project [projectNo=101, title=BookService, description=Book Sale in Online, duration=60, status=Development]
```

SqlUpdate

➔It similar to SqlQuery class but given for non-select queries (update queries) execution.
➔We can prepare subclass to SqlUpdate class inside the DAO class where we supply datasource,query to SqlUpdate class only once.
➔But we use these sub classes object for multiple times to execute same non select query with the support of update() method to execute non-select query with same or different parameters for multiple times.
➔For every non select query we need to take separate sub class for SqlUpdate inside the DAO class.

SQlUpdate-1

```
public class Test {
        public static void main(String[] args) throws SQLException {
                ApplicationContext context=
                                new ClassPathXmlApplicationContext("com/sjf/common/application-context.xml");
                ProjectDAO projectDAO=context.getBean("projectDAO",ProjectDAO.class);
                int rowCount=projectDAO.updateProjects(101, "BookServceInfo");
                System.out.println(rowCount);
        }
}
```

```
public class ProjectDAO {
        private DataSource dataSource;
        private ProjectSqlUpdate psu;
        private final String SQL_FOR_PROJECT = "UPDATE PROJECT  SET TITLE=? WHERE PROJECT_NO=?";

        public ProjectDAO(DataSource dataSource) {
                this.dataSource = dataSource;
                psu = new ProjectSqlUpdate(dataSource, SQL_FOR_PROJECT);
        }

        public int updateProjects(int projectNo, String title) {
                return psu.update(new Object[] { title, projectNo });
        }

        private final class ProjectSqlUpdate extends SqlUpdate {
                public ProjectSqlUpdate(DataSource dataSource, String sqlQuery) {
                        super(dataSource, sqlQuery);
                        declareParameter(new SqlParameter(java.sql.Types.VARCHAR));
                        declareParameter(new SqlParameter(java.sql.Types.INTEGER));
                        compile();
                }
        }
}
```

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

        <bean id="dataSource" 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="spr_usr"/>
                <property name="password" value="welcome1"/>
        </bean>
        <bean id="projectDAO" class="com.sjf.dao.ProjectDAO">
                <constructor-arg ref="dataSource"/>
         </bean>
</beans>
```

SQlUpdate-2

```
public class Test {
	public static void main(String[] args) throws SQLException {
		ApplicationContext context=
				new ClassPathXmlApplicationContext("com/sjf/common/application-context.xml");
		ProjectDAO projectDAO=context.getBean("projectDAO",ProjectDAO.class);
		int rowCount=projectDAO.saveProjects(150, "Villege Management System","VMS",60,"Developlemt");
		System.out.println(rowCount);
	}
}
```

```
public class ProjectDAO {
	private DataSource dataSource;
	private ProjectSqlUpdate psu;
	private final String SQL_FOR_PROJECT = "INSERT INTO PROJECT  VALUES(?,?,?,?,?)";

	public ProjectDAO(DataSource dataSource) {
		this.dataSource = dataSource;
		psu = new ProjectSqlUpdate(dataSource, SQL_FOR_PROJECT);
	}

	public int saveProjects(int projectNo, String title,String description,int duration,String status) {
		return psu.update(new Object[] {projectNo,title,description,duration,status });
	}

	private final class ProjectSqlUpdate extends SqlUpdate {
		public ProjectSqlUpdate(DataSource dataSource, String sqlQuery) {
			super(dataSource, sqlQuery);
			declareParameter(new SqlParameter(java.sql.Types.INTEGER));
			declareParameter(new SqlParameter(java.sql.Types.VARCHAR));
			declareParameter(new SqlParameter(java.sql.Types.VARCHAR));
			declareParameter(new SqlParameter(java.sql.Types.INTEGER));
			declareParameter(new SqlParameter(java.sql.Types.VARCHAR));
			compile();
		}
	}
}
```