Spring Boot - JDBC

Last Updated : 23 Jul, 2025

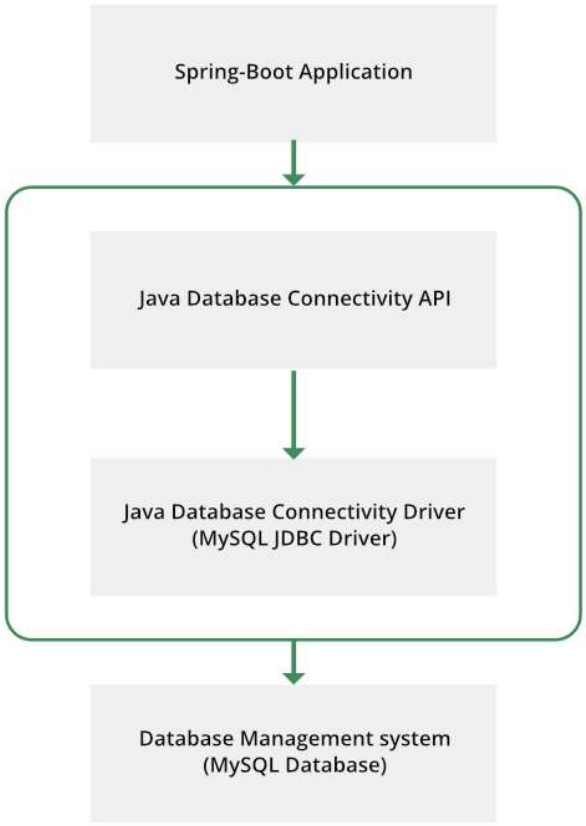
Spring Boot JDBC is used to connect the Spring Boot application with **JDBC** by providing libraries and starter dependencies. Spring Boot JDBC has a level of control over the SQL queries that are being written. Spring Boot JDBC has simplified the work of Spring JDBC by automating the work and steps by using the concept of **Auto Configuration**. **JdbcTemplate**, **DataSource, NamedParameterJdbcTemplate,** etc. all required objects are auto configured, and object creation is done by Spring Boot.

To build an application from scratch, you just need to use JdbcTemplate. As a programmer, you need to auto-wire the **JdbcTemplate** and perform the database operations like **Create, Retrieve, Update, and Delete**. Inputs need to be passed by **application.properties** or **application.yml file.** In application.properties file, we configure **DataSource** and **connection pooling.**

JDBC consists of two parts as depicted below:

- **JDBC interfaces:** *java.sql / javax.sql* packages have classes/interfaces of JDBC API.
- **JDBC drivers:** *JDBC Driver* allows Java programs to interact with the database.

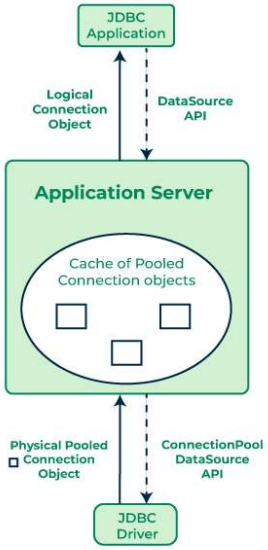
Spring Boot offers many ways to work with databases (e.g - JdbcTemplate) without the cumbersome effort that JDBC needs. You can use raw JDBC to manually configure the workings.



JDBC Connection Pooling

JDBC Connection pooling is a mechanism which makes a database connection as reusable for more than one client, so burden on a database server will be reduced. A connection pool contains set of pooled connections or reusable connections.

When we use DriverManager or DataSource, the connection opened with a DataSource is a non-reusable. For each client if a new connection is opened, the burden on database server will be increase. So, to reduce the burden on database server, we use connection pooling.



How Connection Pooling Works

- A server administrator prepares a connection pool with a setoff connection and also a mediator object as a DataSource API to access the pool.
- An administrator stores DataSource object into JNDI (Java Naming Directory Interface).
- Application reads the DataSource from JNDI registry and asks for a connection from a pool.
- A DataSource object takes a connection from a pool and creates a proxy or logical connection to it and then sens the proxy connection to Java.
- When a Java program closes proxy connection, the DataSource will do real connection back to the pool.

To work with a database using Spring Boot, we need to add the following dependencies:

A. JDBC API

Database connectivity API specifies how the client connects and queries a database.

pom.xml Maven Dependency:

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-jdbc</artifactId>
</dependency>
```

B. MySQL Driver

MySQL JDBC and R2DBC driver to work with the database.

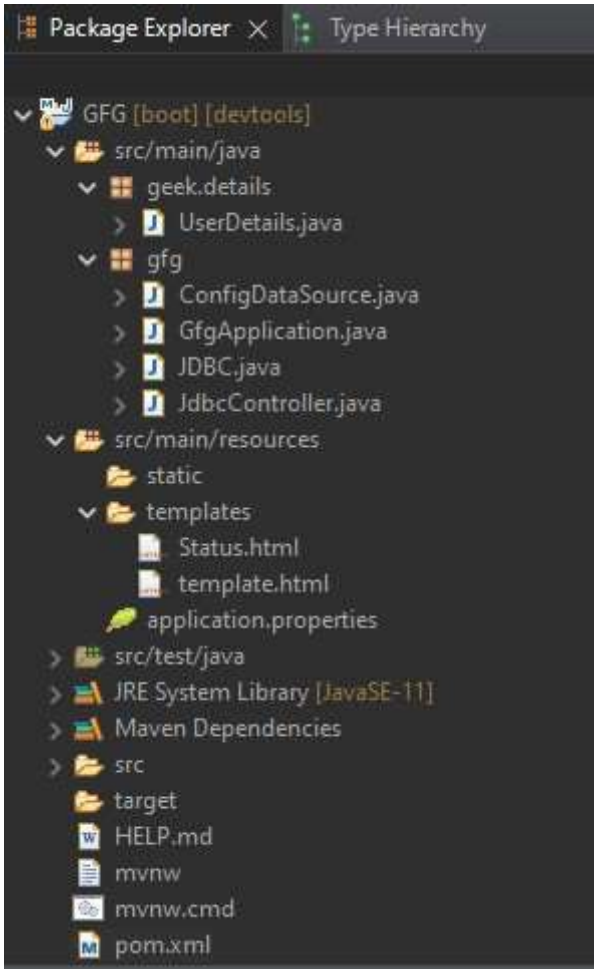
```
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
  <scope>runtime</scope>
</dependency>
```

After this, we will add **datasource** properties in **application.properties** file:

For MySQL database:

```
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:8080/test
spring.datasource.username=root
spring.datasource.password=root
```

Project Structure



Implementation of an Application

A. pom.xml (Configuration File)

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="https://maven.apache.org/POM/4.0.0"
         xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="https://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.6.3</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>sia</groupId>
    <artifactId>GFG</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>GFG</name>
    <description>Demo project for Spring Boot</description>
    <properties>
        <java.version>11</java.version>
    </properties>
    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
```

```
            <artifactId>spring-boot-starter-thymeleaf</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-devtools</artifactId>
            <scope>runtime</scope>
            <optional>true</optional>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter</artifactId>
        </dependency>
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <scope>runtime</scope>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-jdbc</artifactId>
        </dependency>
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <optional>true</optional>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
```



```
            <groupId>org.springframework.security</groupId>
            <artifactId>spring-security-test</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
                <configuration>
                    <excludes>
                        <exclude>
```

```
                        <groupId>org.projectlombok</groupId>
                        <artifactId>lombok</artifactId>
                    </exclude>
                </excludes>
            </configuration>
        </plugin>
    </plugins>
</build>

</project>
```

B. Boot of Spring application (GfgApplication.java)

```
// Java Program to Illustrate Boot of Spring Application

package gfg;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

// Annotation
@SpringBootApplication

// Application(Main) Class
public class GfgApplication {

    // Main driver method
    public static void main(String[] args)
    {

        SpringApplication.run(GfgApplication.class, args);
    }
}
```

C. Configuration of DataSource (ConfigDataSource.java)

DataSourceBuilder<T> is a useful class to build a DataSource.

- org.springframework.boot.jdbc.DataSourceBuilder<T>
- public final class DataSourceBuilder<T extends DataSource> extends Object

Methods of DataSourceBuilder<T> Class

Method	Description
create()	Creates a *new instance* of DataSourceBuilder.
driverClassName(String driverClassName)	Specifies the *driver class name* which is to be used for building the datasource.
url(String url)	Specifies the *URL* which is to be used for building the datasource.
username(String username)	Specifies the *username* which is to be used for building the datasource.
password(String password)	Specifies the *password* which is to be used for building the datasource.
build()	Returns a newly built *DataSource instance*.

This builder supports the following pooling Datasource implementations.

Name	Description
Hikari	com.zaxxer.hikari.HikariDataSource
Tomcat JDBC Pool	org.apache.tomcat.jdbc.pool.DataSource
Apache DBCP2	org.apache.commons.dbcp2.BasicDataSource

Name	Description
Oracle UCP	oracle.ucp.jdbc.PoolDataSourceImpl

Note: *The first available pool implementation is used when no type has been explicitly set.*

```
INFO 1472 --- [nio-8080-exec-2] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Starting...
```

Example:

```
// Java Program Illustrating Configuration of
// DataSourceConfiguration of DataSource

package gfg;

import javax.sql.DataSource;
import org.springframework.boot.jdbc.DataSourceBuilder;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

// Annotation
@Configuration

// Class
public class ConfigDataSource {

    @Bean public static DataSource source()
    {

        DataSourceBuilder<?> dSB
            = DataSourceBuilder.create();
        dSB.driverClassName("com.mysql.jdbc.Driver");

        // MySQL specific url with database name
        dSB.url("jdbc:mysql://localhost:3306/userdetails");

        // MySQL username credential
        dSB.username("user");

        // MySQL password credential
        dSB.password("password");

        return dSB.build();
    }
}
```

> ***Note:*** *Driver class name - 'com.mysql.jdbc.Driver' has been deprecated. It would not give error because the driver is automatically loaded and manual loading is not necessary. The new driver class name is '**com.mysql.cj.jdbc.Driver**'.*

D. User credentials to be stored in the database (UserDetails.java)

One can add the 'Lombok' library to skip Getter/Setter methods, construct No-Arguments constructor, the constructor for final fields, etc.

Maven-pom.xml dependency

```
<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <optional>true</optional>
</dependency>
```

Example:

```
// Java Program Illustrating User Credentials to
// be Stored in the Database

package geek.details;

import lombok.Data;

// Annotation for Getter/Setter methods
@Data
public class UserDetails {

    String user;
    String userName;
    String password;
}
```

E. Utility class for connecting and querying the database (JDBC.java)

Password encoding is a must for many security reasons. To use Spring Security, add the following dependency:

Maven - pom.xml

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
</dependency>

<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-test</artifactId>
    <scope>test</scope>
</dependency>
```

Pre-requisites are as follows:

1. On adding the above dependency, the login page gets activated by default.
2. The default username is - 'user.'
3. The password is automatically generated by the Spring Security that gets displayed on the console after booting the application.

```
Using generated security password: f9424c44-93cf-4443-92b6-bbccdedf75d1
```

> ***Note:*** *Generated password becomes invalid for the next iteration/Running the application as the new password gets generated, but the username remains same.*