



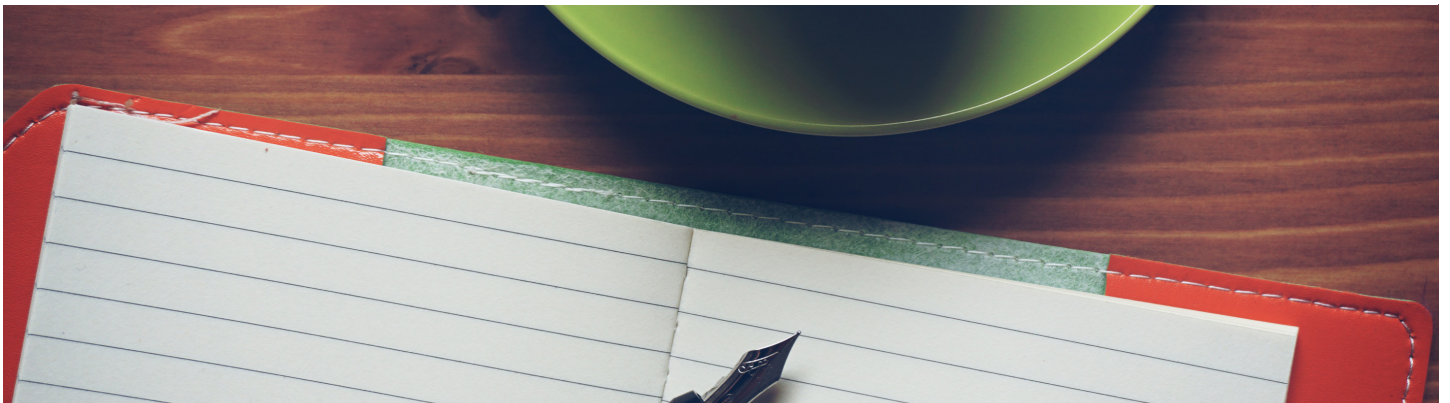
# devs4j

EL MEJOR SITIO WEB SOBRE PROGRAMACIÓN EN ESPAÑOL.

HOME

ABOUT

CONTACT



Anuncios

AUTOMATTIC

**We're hiring PHP developers  
anywhere in the world. Join us!**

APPLY



[Report this ad](#)

# Conecta una aplicación a multiples bases de datos condicionalmente

 HACE 2 SEMANAS     DEJA UN COMENTARIO

1 Vote

Un problema común en aplicaciones complejas, es cuando deseas desarrollar una aplicación que consume multiples bases de datos de acuerdo al país, area, cliente, etc. Una decisión común para resolver este problema es crear diferentes repositorios de código para cada uno, el problema es al momento de mantener el código y actualizaciones al sistema.

En este post explicaremos como podemos contar con un solo código que se pueda conectar a diferentes bases de datos de acuerdo a una configuración.

## Paso 1 Creando bases de datos

El primer paso será crear dos bases de datos, una será utilizada para almacenar usuarios de Estados unidos y la otra para México:

```
1 create database us_users;
2 use us_users;
3 CREATE TABLE USER(
4 USER_ID INTEGER PRIMARY KEY AUTO_INCREMENT
5 USERNAME VARCHAR(100) NOT NULL,
6 PASSWORD VARCHAR(100) NOT NULL
7 );
8 INSERT INTO USER (USERNAME,PASSWORD)VALUES
9 INSERT INTO USER (USERNAME,PASSWORD)VALUES
10 INSERT INTO USER (USERNAME,PASSWORD)VALUES
```

Lo anterior creará una tabla de usuarios para Estados Unidos junto con algunos registros de ejemplo.

```
1 create database mx_users;
2 use mx_users;
3 CREATE TABLE USER(
4 USER_ID INTEGER PRIMARY KEY AUTO_INCREMENT
5 USERNAME VARCHAR(100) NOT NULL,
6 PASSWORD VARCHAR(100) NOT NULL
7 );
8 INSERT INTO USER (USERNAME,PASSWORD)VALUES
9 INSERT INTO USER (USERNAME,PASSWORD)VALUES
10 INSERT INTO USER (USERNAME,PASSWORD)VALUES
```

Lo anterior creará una tabla de usuarios para México junto con algunos registros de ejemplo.

## Paso 2 Configurado el proyecto

El proyecto se creará con Spring boot con las siguientes dependencias:

<https://github.com/raidentrance/spring-multiple-db/blob/master/pom.xml> (<https://github.com/raidentrance/spring-multiple-db/blob/master/pom.xml>)

Con lo anterior tendremos todo lo necesario para trabajar con **spring-mvc** y con **spring-jdbc**.

## Paso 3 Creando la clase aplicación

Una vez que tenemos configurado spring boot tendremos que crear la clase aplicación, que será quien inicie nuestra api.

```

1  import org.springframework.boot.SpringApplication
2  import org.springframework.boot.autoconfigure
3  import org.springframework.boot.builder.SpringApplicationBuilder
4  import org.springframework.boot.web.servlet.support.SpringBootServletInitializer
5
6  /**
7   * @author raidentrance
8   *
9   */
10 @SpringBootApplication
11 public class SampleApplication extends SpringBootServletInitializer {
12     @Override
13     protected SpringApplicationBuilder configure(SpringApplicationBuilder application) {
14         return application.sources(SampleApplication.class);
15     }
16
17     public static void main(String[] args) {
18         SpringApplication.run(SampleApplication.class, args);
19     }
20
21 }

```

Para ejecutar nuestra aplicación solo ejecutaremos la clase anterior y los servicios se expondrán de forma exitosa.

## Paso 4 Configurando datasources

El siguiente paso será configurar nuestros datasources, para esto crearemos un paquete llamado **com.devs4j.example.config.db** con la siguiente clase:

```

1  import javax.sql.DataSource;
2
3  import org.springframework.boot.autoconfigure.jdbc.DataSourceProperties;
4  import org.springframework.boot.jdbc.DataSourceBuilder;
5  import org.springframework.context.annotation.Bean;
6  import org.springframework.context.annotation.Configuration;
7
8  /**
9   * @author raidentrance
10  *
11  */
12 @Configuration
13 public class DataSourceConfig {
14     @Bean
15     @ConditionalOnProperty(name = "market")
16     public DataSource mxDatasource() {
17         return DataSourceBuilder.create().
18             .url("jdbc:mysql://localhost:3306/mx")
19             .build();
20     }
21
22     @Bean
23     @ConditionalOnProperty(name = "market")
24     public DataSource usDatasource() {

```

```

25         return DataSourceBuilder.create().
26             .url("jdbc:mysql://localhost:3306/");
27     }
28
29 }

```

Como se puede ver se crearon 2 datasources cada uno apuntando a una base de datos diferente, como se puede ver se hace uso de **@ConditionalOnProperty**, esto significa que creará el bean siempre y cuando se cumpla con la condición que define, que en este caso es que el valor de la propiedad market sea igual a us o mx. Otro punto importante es que solo un bean se creará, en caso contrario Spring no sabría que bean inyectar.

## Paso 5 Creando el modelo de nuestra aplicación

Una vez que creamos la tabla, el siguiente paso será crear una clase que la represente, para esto crearemos la siguiente clase:

```

1  /**
2   * @author raidentrance
3   *
4   */
5  public class User {
6      private Integer id;
7      private String username;
8      private String password;
9
10     public User() {
11     }
12
13     public User(Integer id, String username, String password) {
14         super();
15         this.id = id;
16         this.username = username;
17         this.password = password;
18     }
19
20     public Integer getId() {
21         return id;
22     }
23
24     public void setId(Integer id) {
25         this.id = id;
26     }
27
28     public String getUsername() {
29         return username;
30     }
31
32     public void setUsername(String username) {

```

```
33         this.username = username;
34     }
35
36     public String getPassword() {
37         return password;
38     }
39
40     public void setPassword(String password) {
41         this.password = password;
42     }
43
44 }
```

Se creará un objeto de la clase User por cada registro que se desee devolver.

## Paso 6 Creando un DAO de spring jdbc

El siguiente paso será hacer uso de Spring jdbc para ejecutar una consulta a nuestra base de datos, para esto crearemos el siguiente DAO (Data access object):

```
1  import java.sql.ResultSet;
2  import java.sql.SQLException;
3  import java.util.List;
4
5  import org.springframework.beans.factory.annotation.Autowired;
6  import org.springframework.jdbc.core.JdbcTemplate;
7  import org.springframework.jdbc.core.RowMapper;
8  import org.springframework.stereotype.Component;
9
10 import com.devs4j.example.model.User;
11
12 /**
13  * @author raidentrance
14  *
15  */
16 @Component
17 public class UserDao {
18     @Autowired
19     private JdbcTemplate jdbcTemplate;
20
21     private static final String GET_ALL =
22
23     public List getUsers() {
24         return jdbcTemplate.query(GET_ALL,
25
26             @Override
27             public User mapRow(ResultSet rs, int rowNum) throws SQLException {
28                 return new User(rs.getInt("id"), rs.getString("username"), rs.getString("password"));
29             }
30     });
31 }
```

```
32 | }
```

Nuestro DAO solo cuenta con un método que devuelve todos los usuarios en la tabla.

## Paso 7 Creando un Service de spring

En este ejemplo no es tan necesario, pero lo crearemos para que quede clara la capa de servicios en nuestro servicio:

```
1 | import java.util.List;
2 |
3 | import org.springframework.beans.factory.annotation.Autowired;
4 | import org.springframework.stereotype.Service;
5 |
6 | import com.devs4j.example.dao.UserDao;
7 | import com.devs4j.example.model.User;
8 |
9 | /**
10 |  * @author raidentrance
11 |  */
12 |
13 | @Service
14 | public class UserService {
15 |     @Autowired
16 |     private UserDao dao;
17 |
18 |     public List getUsers() {
19 |         return dao.getUsers();
20 |     }
21 | }
```

El servicio UserService utilizará al DAO creado previamente para obtener la información de los usuarios a devolver.

## Paso 8 Creando el Controller

Una vez hecho lo anterior, el siguiente paso será crear un controller que expondrá la información obtenida vía HTTP a través de un servicio REST:

```
1 | import java.util.List;
2 |
3 | import org.springframework.beans.factory.annotation.Autowired;
4 | import org.springframework.http.HttpStatus;
5 | import org.springframework.http.ResponseEntity;
6 | import org.springframework.web.bind.annotation.RequestMapping;
7 | import org.springframework.web.bind.annotation.RequestMethod;
8 | import org.springframework.web.bind.annotation.RestController;
```

```
9
10 import com.devs4j.example.model.User;
11 import com.devs4j.example.service.UserService;
12
13 /**
14  * @author raidentrance
15  *
16  */
17 @RestController
18 @RequestMapping("/api")
19 public class UserController {
20     @Autowired
21     private UserService service;
22
23     @RequestMapping("/users")
24     @ResponseBody
25     public ResponseEntity getUsers() {
26         return new ResponseEntity(service.
27     }
28 }
```

Como se puede ver se expone un endpoint GET /api/users que devolverá la lista de usuarios.

## Paso 9 Creando nuestro archivo application.properties

Por último crearemos un archivo llamado **application.properties** en el folder **/src/main/resources** con lo siguiente:

```
1 | market=mx
```

Lo anterior definirá el mercado que utilizará la aplicación, en este caso será México.

## Paso 10 Probando la aplicación

Una vez que tenemos todo lo anterior, el último paso será probar el api, para esto ejecutaremos nuestra aplicación (recordando que el mercado que definimos fue México) y abriremos la url <http://localhost:8080/api/users> (<http://localhost:8080/api/users>) con la siguiente salida:

```
1 | [
2 |   {
3 |     "id": 5,
4 |     "username": "Alejandro",
5 |     "password": "superSecreta"
```



```
6      },
7      {
8          "id": 6,
9          "username": "Pedro",
10         "password": "Pablito"
11     },
12     {
13         "id": 7,
14         "username": "Pancho",
15         "password": "Pantera"
16     }
17 ]
```

Como podemos ver los usuarios que se muestran son los que definimos en la base de datos de México, ahora modificaremos el archivo **application.properties** con lo siguiente:

```
1 | market=us
```

Una vez hecho esto ejecutaremos nuevamente la aplicación e invocaremos de nuevo la url <http://localhost:8080/api/users> (<http://localhost:8080/api/users>) con la siguiente salida:

```
1  [
2      {
3          "id": 1,
4          "username": "emma",
5          "password": "superSecret"
6      },
7      {
8          "id": 2,
9          "username": "john",
10         "password": "smith"
11     },
12     {
13         "id": 3,
14         "username": "kc",
15         "password": "helloworld"
16     }
17 ]
```

Como vemos ahora estamos obteniendo la información de la base de datos de Estados Unidos.

Puedes encontrar el código completo en el siguiente

link <https://github.com/raidentrance/spring-multiple-db/blob/master/pom.xml> (<https://github.com/raidentrance/spring-multiple-db/blob/master/pom.xml>).

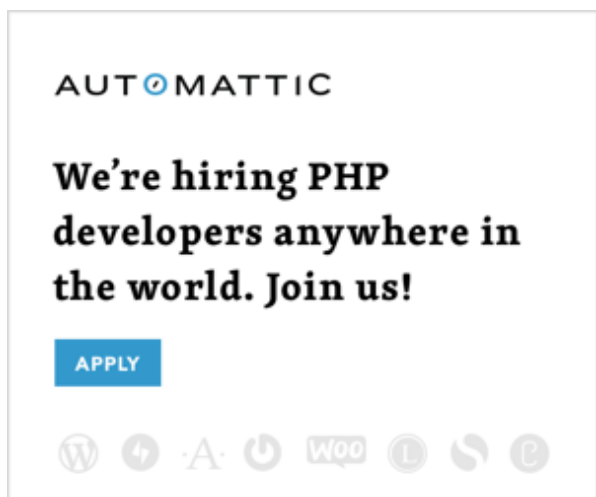
Si te gusta el contenido y quieres enterarte cuando realicemos un post nuevo síguenos en nuestras redes sociales [https://twitter.com/geeks\\_mx](https://twitter.com/geeks_mx) ([https://twitter.com/geeks\\_mx](https://twitter.com/geeks_mx)) y <https://www.facebook.com/geeksJavaMexico/> (<https://www.facebook.com/geeksJavaMexico/>).

*Autor: Alejandro Agapito Bautista*

*Twitter: @raidentrance (<https://twitter.com/raidentrance>)*

*Contacto: [raidentrance@gmail.com](mailto:raidentrance@gmail.com)*

#### Anuncios



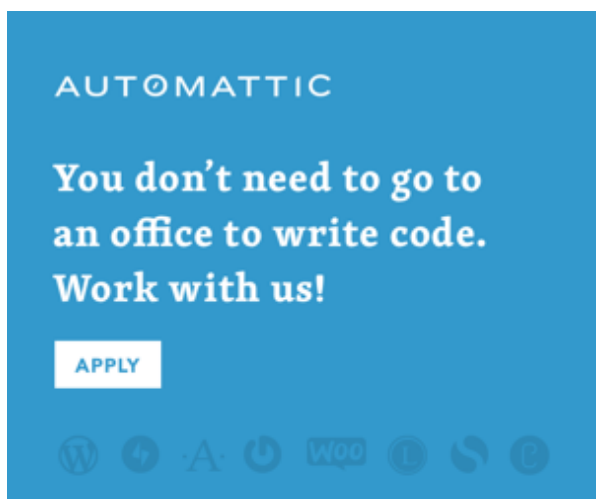
AUTOMATTIC

**We're hiring PHP developers anywhere in the world. Join us!**

APPLY

Logos for various technologies: WordPress, Laravel, Angular, React, WooCommerce, Drupal, Joomla, Magento.

[Report this ad](#)



AUTOMATTIC

**You don't need to go to an office to write code. Work with us!**

APPLY

Logos for various technologies: WordPress, Laravel, Angular, React, WooCommerce, Drupal, Joomla, Magento.

[Report this ad](#)

