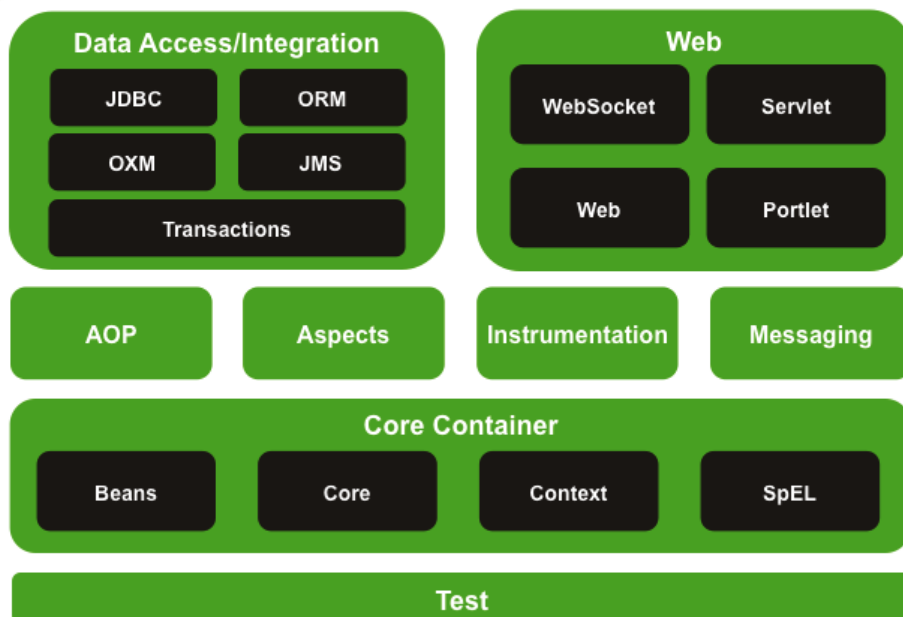




DESARROLLO WEB CON **SPRING BOOT**



Spring Framework Runtime



UNIDAD 07

PARTE B - SPRING DATA JDBC

Eric Gustavo Coronel Castillo

www.desarrollasoftware.com

gcoronelc@gmail.com

I N S T R U C T O R

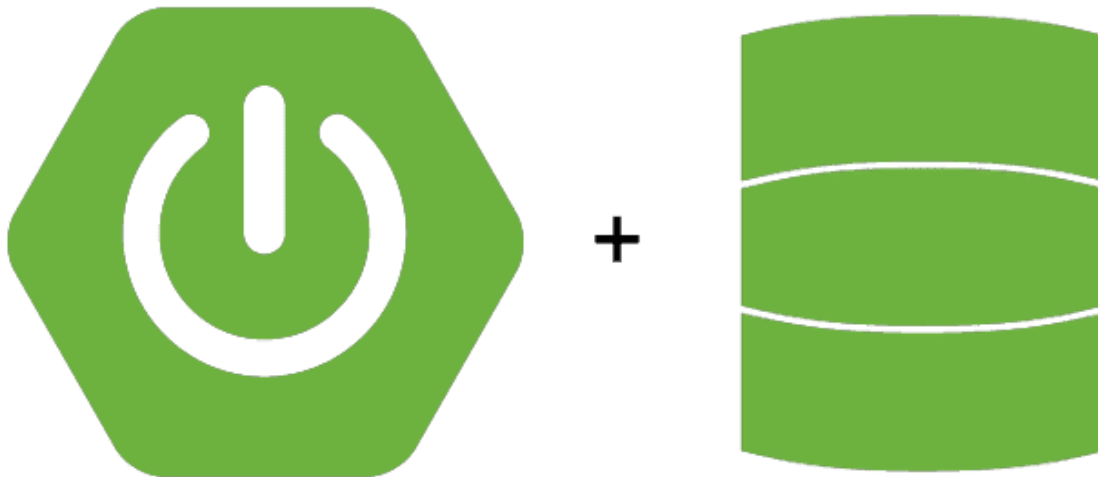


CONTENIDO

INTRODUCCIÓN	3
CONFIGURACIÓN	4
CREACIÓN DEL PROYECTO	4
DEPENDENCIAS	4
PARÁMETROS DE CONEXIÓN	5
IMPLEMENTACIÓN DEL REPOSITORY	6
LA CLASE MODEL	6
LA CAPA REPOSITORY	7
LA CAPA SERVICE	7
PRUEBA DE LA CAPA SERVICE	8
CREANDO CONSULTAS	9
IMPLEMENTANDO LA CONSULTA	9
IMPLEMENTANDO LA CAPA SERVICE	9
PROBANDO LA CAPA SERVICE	9
PERSONALIZANDO LA CONSULTA	10
REDEFINIENDO LA CONSULTA	10
REDEFINIENDO EL SERVICE	10
PROBANDO EL SERVICE	11
CURSOS VIRTUALES	12
CUPONES	12
FUNDAMENTOS DE PROGRAMACIÓN CON JAVA	12
JAVA ORIENTADO A OBJETOS	13
PROGRAMACIÓN CON JAVA JDBC	14
PROGRAMACIÓN CON ORACLE PL/SQL	15



INTRODUCCIÓN



El objetivo principal del proyecto Spring Data es facilitar la creación de aplicaciones impulsadas por Spring que utilizan nuevas tecnologías de acceso a datos, como bases de datos no relacionales y servicios de datos basados en la nube.

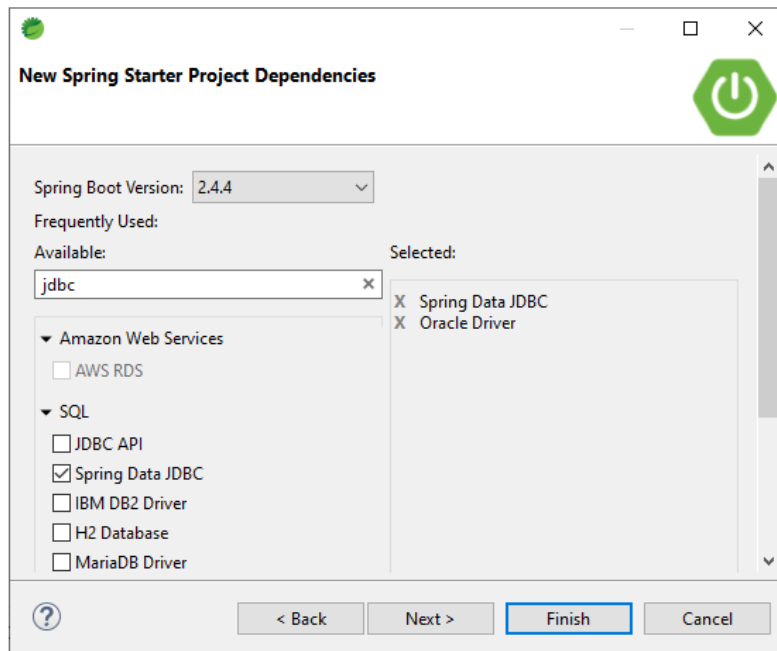
Spring Data JDBC, parte de la familia Spring Data más grande, facilita la implementación de repositorios basados en JDBC. Este módulo se ocupa de la compatibilidad mejorada para las capas de acceso a datos basadas en JDBC. Facilita la creación de aplicaciones impulsadas por Spring que utilizan tecnologías de acceso a datos.

Se encuentra más cerca de la base de datos porque no contiene la mayor parte de la magia de Spring Data al consultar la base de datos. Cada consulta se ejecuta directamente en el JDBC y no hay carga diferida ni almacenamiento en caché, por lo que cuando consulta la base de datos, devolverá el resultado completo.



CONFIGURACIÓN

Creación del proyecto



Dependencias

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

<dependency>
  <groupId>com.oracle.database.jdbc</groupId>
  <artifactId>ojdbc8</artifactId>
  <scope>runtime</scope>
</dependency>
```



Parámetros de Conexión

Estos parámetros se deben incluir en el archivo de propiedades (application.properties), a continuación tienes los parámetros con Oracle Database.

```
# Oracle settings
spring.datasource.url=jdbc:oracle:thin:@<servidor>:1521/<servicio>
spring.datasource.username=<usuario>
spring.datasource.password=<clave>
```

Este es un ejemplo:

```
# Oracle settings
spring.datasource.url=jdbc:oracle:thin:@localhost:1521/XE
spring.datasource.username=eureka
spring.datasource.password=admin

logging.level.=ERROR
logging.level.org.springframework.jdbc.core = TRACE
```



IMPLEMENTACIÓN DEL REPOSITORY

La Clase Model

```
@Table("CLIENTE")
public class Cliente implements Serializable {

    private static final long serialVersionUID = 1L;

    @Id
    @Column("CHR_CLIECODIGO")
    private String codigo;

    @Column("VCH_CLIEPATERNO")
    private String paterno;

    @Column("VCH_CLIEMATERNO")
    private String materno;

    @Column("VCH_CLIENOMBRE")
    private String nombre;

    @Column("CHR_CLIEDNI")
    private String dni;

    @Column("VCH_CLIECIUDAD")
    private String ciudad;

    @Column("VCH_CLIEDIRECCION")
    private String direccion;

    @Column("VCH_CLIETELEFONO")
    private String telefono;

    @Column("VCH_CLIEEMAIL")
    private String email;

    public Cliente() {
    }

    // Métodos setters y getters

}
```



La Capa Repository

```
@Repository
public interface ClienteRepository extends CrudRepository<Cliente, String> {

}
```

La Capa Service

```
@Service
public class ClienteService {

    @Autowired
    private ClienteRepository clienteRepository;

    public Cliente insertar(Cliente cliente) {
        return clienteRepository.save(cliente);
    }

}
```

En este caso, para que funcione la inserción de nuevos clientes, se debe implementar el siguiente disparador:

```
CREATE OR REPLACE TRIGGER EUREKA.TR_CLIENTE_GENERA_CODIGO
BEFORE INSERT ON EUREKA.CLIENTE
FOR EACH ROW
DECLARE
    V_ITEM NUMBER(10,0);
    V_LONG NUMBER(10,0);
    V_CODIGO VARCHAR(20);
BEGIN
    IF INSERTING AND :NEW.CHR_CLIECODIGO IS NULL THEN
        UPDATE EUREKA.CONTADOR
            SET int_contitem = int_contitem + 1
            WHERE UPPER(vch_conttabla) = 'CLIENTE'
            RETURNING int_contitem, int_contlongitud INTO V_ITEM, V_LONG;
```



```
V_CODIGO := TO_CHAR(V_ITEM);  
V_CODIGO := LPAD(V_CODIGO,V_LONG,'0');  
:NEW.CHR_CLIECODIGO := V_CODIGO;  
END IF;  
END TR_CLIENTE_GENERA_CODIGO;  
/
```

Prueba de la Capa Service

```
Cliente cliente = new Cliente();  
cliente.setCodigo(null);  
cliente.setPaterno("Torres");  
cliente.setMaterno("Castro");  
cliente.setNombre("Juan");  
cliente.setDni("12345678");  
cliente.setDireccion("Lima");  
cliente.setCiudad("Lima");  
cliente.setEmail("juan@gmail.com");  
cliente = clienteService.insertar(cliente);  
System.out.println("Código: " + cliente.getCodigo());
```




CREANDO CONSULTAS

Implementando la consulta

```
@Repository
public interface ClienteRepository extends CrudRepository<Cliente, String> {

    List<Cliente> findByPaternoAndMaterno(String paterno, String materno);

}
```

Implementando la capa service

```
@Service
public class ClienteService {

    @Autowired
    private ClienteRepository clienteRepository;

    public Cliente insertar(Cliente cliente) {
        return clienteRepository.save(cliente);
    }

    public List<Cliente> findByPaternoAndMaterno(String paterno, String materno){
        return clienteRepository.findByPaternoAndMaterno(paterno, materno);
    }

}
```

Probando la capa service

```
List<Cliente> lista;
Lista = clienteService.findByPaternoAndMaterno("CORONEL", "CASTILLO");
for(Cliente r: lista) {
    System.out.println(r.getCodigo() + " - "
        + r.getPaterno() + " - " + r.getMaterno());
}
```



PERSONALIZANDO LA CONSULTA

Redefiniendo la consulta

```
@Repository
public interface ClienteRepository extends CrudRepository<Cliente, String> {

    @Query("SELECT * FROM eureka.cliente "
        + "WHERE upper(vch_cliopaterno) like upper(:paterno) "
        + "AND upper(vch_cliomaterno) like upper(:materno) ")
    List<Cliente> findByPaternoAndMaterno(@Param("paterno") String paterno,
        @Param("materno") String materno);

}
```

Redefiniendo el service

```
@Service
public class ClienteService {

    @Autowired
    private ClienteRepository clienteRepository;

    public Cliente insertar(Cliente cliente) {
        return clienteRepository.save(cliente);
    }

    public List<Cliente> findByPaternoAndMaterno(String paterno, String materno){
        paterno = "%" + paterno + "%";
        materno = "%" + materno + "%";
        return clienteRepository.findByPaternoAndMaterno(paterno, materno);
    }

}
```



Probando el service

```
List<Cliente> lista = clienteService.findByPaternoAndMaterno("ar", "");  
for(Cliente r: lista) {  
    System.out.println(r.getCodigo() + " - "  
        + r.getPaterno() + " - " + r.getMaterno());  
}
```



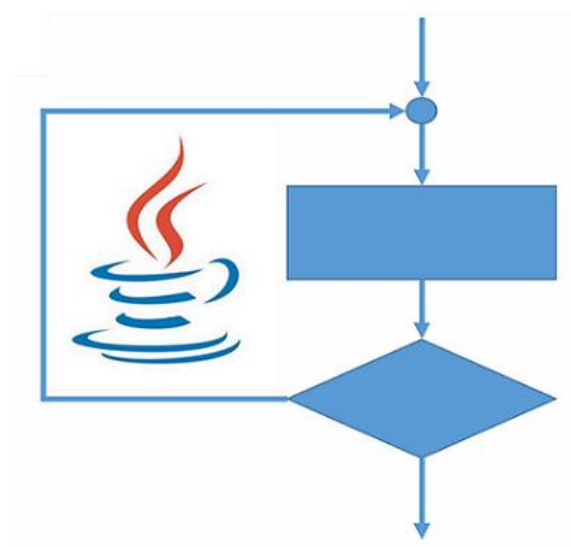
CURSOS VIRTUALES

Cupones

En esta URL se publican cupones de descuento:

<http://gcoronelc.github.io>

Fundamentos de Programación con Java



Tener bases sólidas de programación muchas veces no es fácil, creo que es principalmente por que en algún momento de tu aprendizaje mezclas la entrada de datos con el proceso de los mismos, o mezclas el proceso con la salida o reporte, esto te lleva a utilizar malas prácticas de programación que luego te serán muy difíciles de superar.

En este curso aprenderás las mejores prácticas de programación para que te inicies con éxito en este competitivo mundo del desarrollo de software.

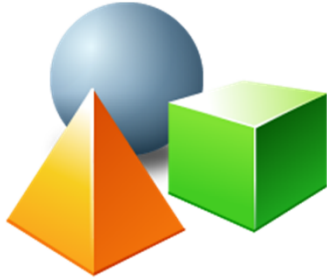
URL del Curso: <https://n9.cl/gcoronelc-java-fund>

Avance del curso: <https://n9.cl/gcoronelc-fp-avance>

Cupones de descuento: <http://gcoronelc.github.io>



Java Orientado a Objetos



CURSO PROFESIONAL DE JAVA ORIENTADO A OBJETOS

Eric Gustavo Coronel Castillo

www.desarrollasoftware.com

I N S T R U C T O R

En este curso aprenderás a crear software aplicando la Orientación a Objetos, la programación en capas, el uso de patrones de software y Swing.

Cada tema está desarrollado con ejemplos que demuestran los conceptos teóricos y finalizan con un proyecto aplicativo.

URL del Curso: <https://bit.ly/2B3ixUW>

Avance del curso: <https://bit.ly/2RYGXIt>

Cupones de descuento: <http://gcoronelc.github.io>



Programación con Java JDBC



PROGRAMACIÓN DE BASE DE DATOS ORACLE CON JAVA JDBC

Eric Gustavo Coronel Castillo

www.desarrollasoftware.com

I N S T R U C T O R

En este curso aprenderás a programar bases de datos Oracle con JDBC utilizando los objetos Statement, PreparedStatement, CallableStatement y a programar transacciones correctamente teniendo en cuenta su rendimiento y concurrencia.

Al final del curso se integra todo lo desarrollado en una aplicación de escritorio.

URL del Curso: <https://bit.ly/31apy0O>

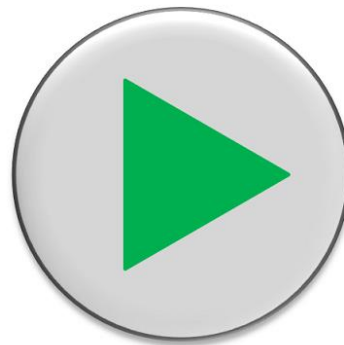
Avance del curso: <https://bit.ly/2vatZOT>

Cupones de descuento: <http://gcoronelc.github.io>



Programación con Oracle PL/SQL

ORACLE PL/SQL



En este curso aprenderás a programar las bases de datos ORACLE con PL/SQL, de esta manera estarás aprovechando las ventajas que brinda este motor de base de datos y mejorarás el rendimiento de tus consultas, transacciones y la concurrencia.

Los procedimientos almacenados que desarrolles con PL/SQL se pueden ejecutar de Java, C#, PHP y otros lenguajes de programación.

URL del Curso: <https://bit.ly/2YZjfxT>

Avance del curso: <https://bit.ly/3bcqYb>

Cupones de descuento: <http://gcoronelc.github.io>