Spring Boot

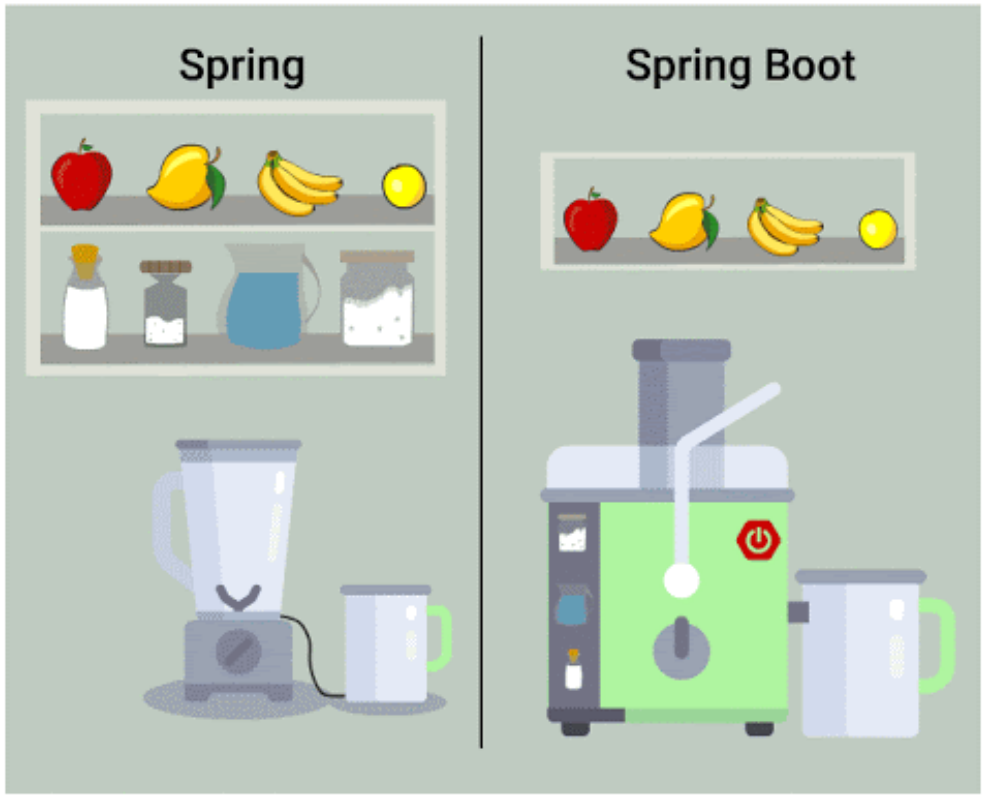
Agenda

- Que es Spring Boot
- Repositorios Spring Data
- JPA

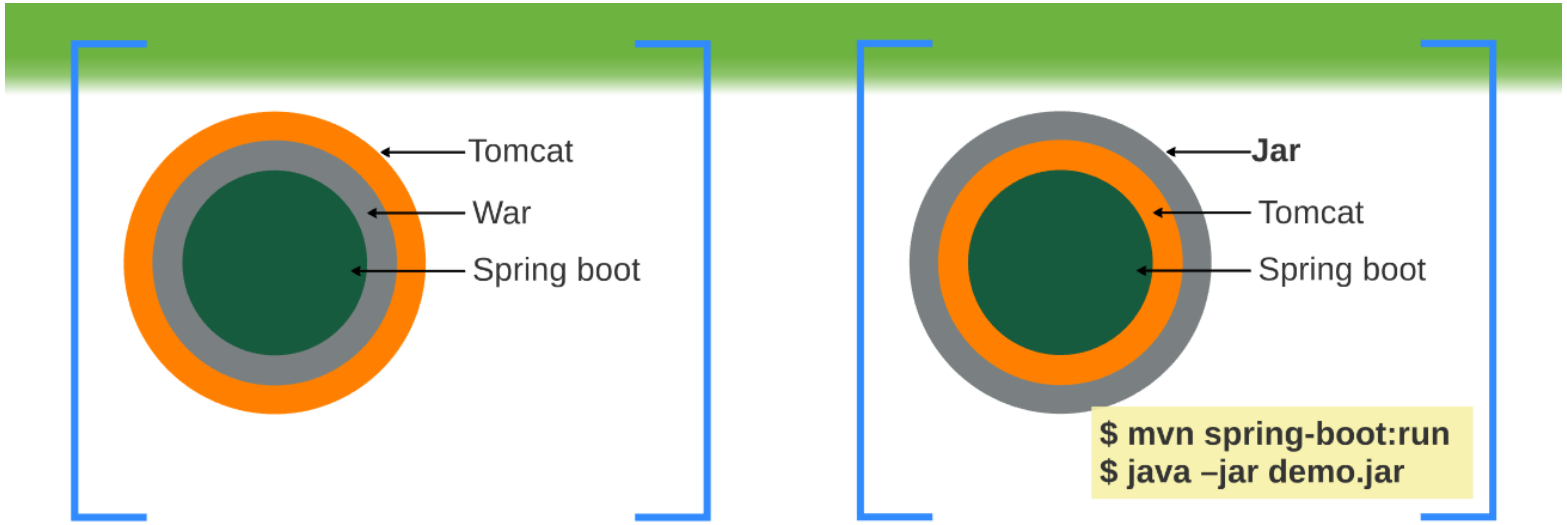
Spring Boot

- Spring es un framework para el desarrollo de aplicaciones web y microservicios.
- Contenedor de inversión de control, de código abierto para la plataforma Java.
- Es un Framework basado en XML ocultado por las anotaciones.
- Altamente configurable
- Trae las dependencias necesarias para un tipo de aplicación
- Soporta todas la librerías de terceros, No SQL DB, Distributed Cache, JPA, REST, etc.

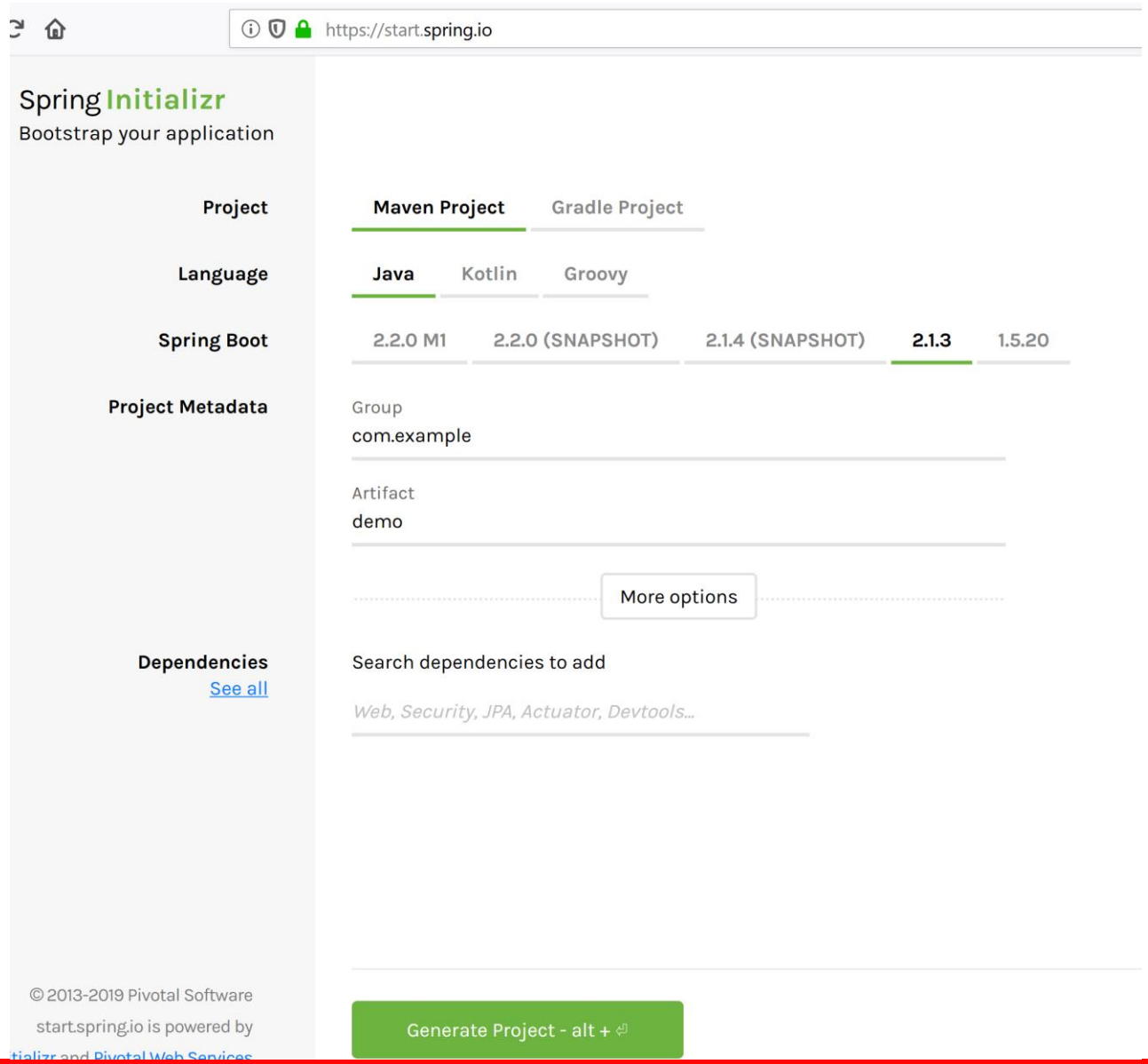
- Embebed Application Server Tomcat
- Integración con Herramientas y Tecnologías
- Herramientas de producción, monitoreo
- Manejo de configuración
- DevTools, hot deployment ante cambios
- No XML, No generación de Código Fuente


Spring
Spring Boot

Como está compuesto



http://start.spring.io



JPA
Java Persistent API

@UPC

Agenda

ORM

ORM-JPA

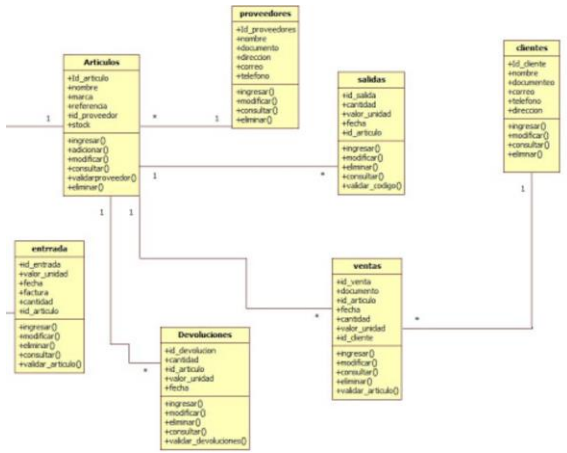
Persistence Provider

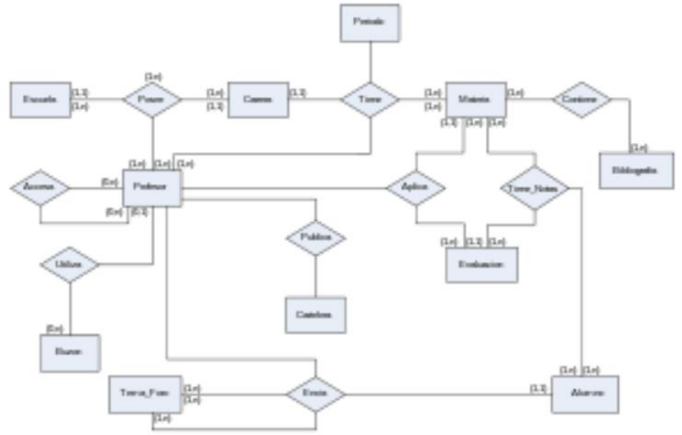
Entitiy Manager

JQL

ORM

Object Relational Mapping, es una técnica de programación para convertir datos entre bases de datos relacional y objetos java.





ORM - JPA

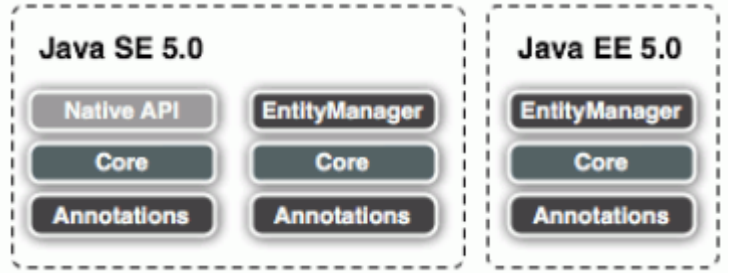
- El proceso de asignación de objetos Java a tablas de bases de datos y viceversa se llama "mapeo objeto-relacional" (ORM).
- El Java Persistence API (JPA) es un método para ORM. A través de JPA el desarrollador puede asignar, almacenar, actualizar y recuperar datos de bases de datos relacionales a objetos Java y viceversa
- JPA permite al desarrollador trabajar directamente con los objetos en lugar de sentencias SQL.
- JPA tiene varias implementaciones disponibles: Eclipse Link, Hibernate, TopLink, etc.

Entidades-> JPA

Entity beans son deprecados a partir de JEE5

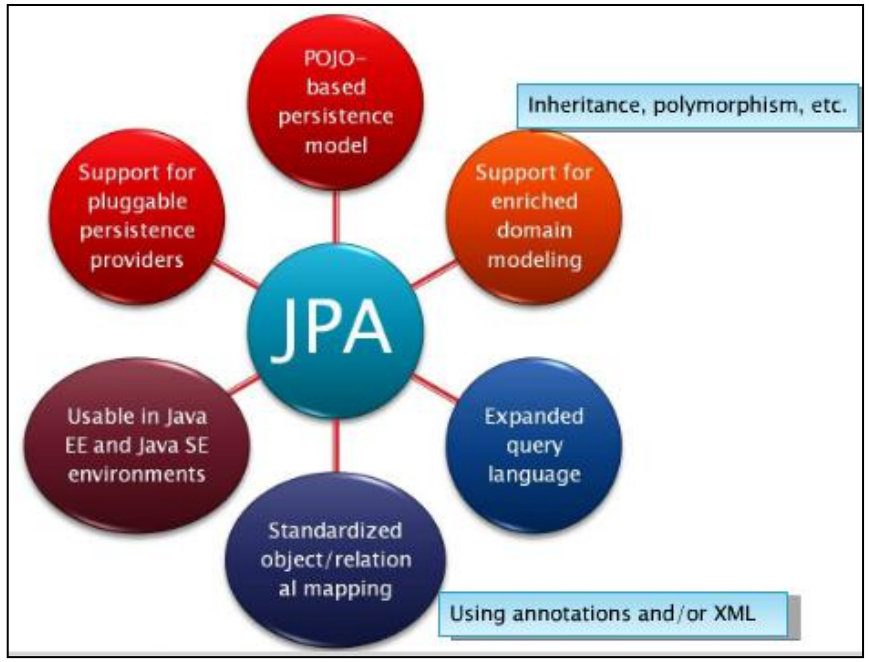
Y ahora hay un nuevo estándar:

- **JPA** – Java Persistence API
- No Application Server
- Provides full O/R mapping
- JPA: Es un estándar ORM e interfaces de manejo de persistencia desde Java EE 5.0 y soportado por la mayoría de vendedores:



JPA

- Basado en POJO
- No requiere interfaces o subclases
- Basado en Annotaciones - Opcional DD
- Soportan modelos de herencia
- EJBQL extendido, JPAQL



Proceso de desarrollo

Top-down: Comenzamos con el desarrollo del modelo de entidades Java y el fichero de mapeo, y generamos el script de la BBDD a partir de estos dos componentes (hbm2dll).

Bottom-up: Se parte de un modelo de datos ER y mediante la utilidad hbm2hbmxml se generan las clases java mapeadas.

Persistence Mapping

O/R Mapping Metadata como annotations

- Table mappings: @Table

Column mappings: @Column

Identifier (Primary-Key): @Id

Relationships:

- @ManyToOne, @OneToOne,
- @OneToMany, @ManyToMany

Inheritance: @Inheritance

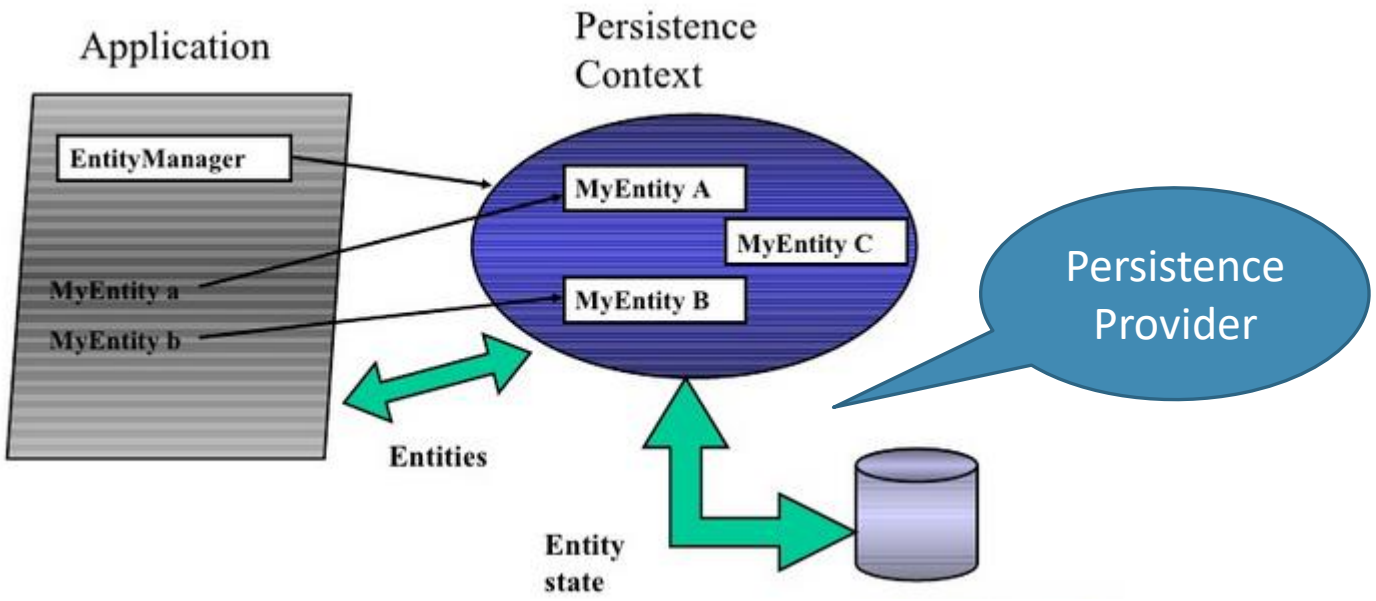
Ejemplo

```
create table EMPLOYEE_TP
(
   EMPLOYEE_ID Number NOT NULL AUTO_INCREMENT,
    EMPLOYEE_NAME varchar(50),
    SALARY Number,
    PRIMARY KEY (‘ÉMPLOYEE_ID’),
);
```

```
@Entity
@Table(name="EMPLOYEE_TP")
public class Employee{
   @Id @GeneratedValue
   @Column(name="EMPLOYEE_ID")
   private int id;
   @Column(name="EMPLOYEE_NAME")
   private String name;
   @Column(name="SALARY")
   private int salary;

   public int getId() {
      return id;
   }
   public void setId(int id) {
   this.id = id;
   }
    ……
```

Persistence Provider – Entity Manager



JPAQL, HQL

CRUD

```
Employee employee = new Employee("Samuel", "Joseph", "Wurzelbacher");
em.getTransaction().begin();
em.persist(employee);
em.getTransaction().commit();
```

```
Employee employee = new Employee("Samuel", "Joseph", "Wurzelbacher");
Address address = new Address("Holland", "Ohio");
employee.setAddress(address);

em.getTransaction().begin();
em.persist(employee);
em.getTransaction().commit();
```

```
Employee employee = em.find(Employee.class, 1);
```

```
Employee employee = em.find(Employee.class, 1);

em.getTransaction().begin();
employee.setNickname("Joe the Plumber");
em.getTransaction().commit();
```

```
Employee employee = em.find(Employee.class, 1);

em.getTransaction().begin();
em.remove(employee);
em.getTransaction().commit();
```

```
SELECT OBJECT( e ) FROM Employee AS e
```

```
SELECT e.department
FROM Employee e
```

```
SELECT e
FROM Employee e
WHERE e.department.name = 'NA42' AND
      e.address.state IN ('NY','CA')
```

```
SELECT e
FROM Employee e
WHERE e.department.name = 'NA42' AND
      e.address.state IN ('NY','CA')
```

```
SELECT p.number
FROM Employee e, Phone p
WHERE e = p.employee AND
      e.department.name = 'NA42' AND
      p.type = 'Cell'
```

```
SELECT e
FROM Employee e
WHERE e.department = :dept AND
      e.salary > :base
```

```
SELECT p.number
FROM Employee e JOIN e.phones p
WHERE e.department.name = 'NA42' AND
      p.type = 'Cell'
```

```
SELECT d, COUNT(e), MAX(e.salary), AVG(e.salary)
FROM Department d JOIN d.employees e
GROUP BY d
HAVING COUNT(e) >= 5
```

```
SELECT DISTINCT d
FROM Department d, Employee e
WHERE d = e.department
```

```
SELECT d
FROM Employee e JOIN e.department d
```

```
SELECT e
FROM Employee e
WHERE e.department = ?1 AND
      e.salary > ?2
```

Repositorio

SPRING DATA

No más DAO

El objetivo del repositorio de Spring Data es reducir significativamente la cantidad de código repetitivo requerido para implementar capas de acceso a datos para varios tipos de persistencia.

CrudRepsitory prove operaciones CRUD genericas sobre un repositorio
Para un tipo especifico de Entidad

```
public interface CrudRepository<T, ID extends Serializable>
    extends Repository<T, ID> {

    <S extends T> S save(S entity);

    T findOne(ID primaryKey);

    Iterable<T> findAll();

    Long count();

    void delete(T entity);

    boolean exists(ID primaryKey);

    // … more functionality omitted.
}
```

Su uso:

Solo inyectando en la clase que lo necesita como una propiedad privada:

```
@Autowire
private ArticleRepository articleRepository;
```

@Transactional con CrudRepository

Es por defecto true.

Se puede configurar:

```
public interface ArticleRepository extends CrudRepository<Article, Long>  {
    @Override
    @Transactional(timeout = 8)
    Iterable<Article> findAll();
}
```

Pasos a Seguir

```java
@Entity
@Table(name="articles")
public class Article implements Serializable {
	private static final long serialVersionUID = 1L;
	@Id
	@GeneratedValue(strategy=GenerationType.AUTO)
	@Column(name="article_id")
	private long articleId;
	@Column(name="title")
	private String title;
	@Column(name="category")
	private String category;
	public long getArticleId() {
		return articleId;
	}
	public void setArticleId(long articleId) {
		this.articleId = articleId;
	}
	public String getTitle() {
		return title;
	}
	public void setTitle(String title) {
		this.title = title;
	}
	public String getCategory() {
		return category;
	}
	public void setCategory(String category) {
		this.category = category;
	}
}
```

Repositorio

```
public interface ArticleRepository extends CrudRepository<Article, Long>  {  }
```

Ejemplo: Si a entidad tiene la propiedad “title” y sus métodos estándar *getTitle y setTitle*, podríamos definer el método ***findByTitle en la interface DAO***; esto deberá generar el correcto query automáticamente:

```
import java.util.List;
import org.springframework.data.repository.CrudRepository;
import com.concretepage.entity.Article;
public interface ArticleRepository extends CrudRepository<Article, Long>  {
    List<Article> findByTitle(String title);
    List<Article> findDistinctByCategory(String category);
    List<Article> findByTitleAndCategory(String title, String category);
}
```

Detalle

```
public interface UserRepository extends Repository<User, Long> {

  List<User> findByEmailAddressAndLastname(String emailAddress, String lastname);
}
```

Crearía automáticamente:

```
select u from User u where u.emailAddress = ?1 and u.lastname = ?2
```

Ejemplo: Custom Repository

```
public interface ArticleRepository extends CrudRepository<Article, Long>  {
    @Query("SELECT a FROM Article a WHERE a.title=:title and a.category=:category")
    List<Article> fetchArticles(@Param("title") String title, @Param("category") String category);
}
```

```java
@Service
public class ArticleService {

    @Autowired
    private ArticleRepository articleRepository;


    public Article getArticleById(long articleId) {
        Article obj = articleRepository.findById(articleId).get();
        return obj;
    }

    public List<Article> getAllArticles(){
        List<Article> list = new ArrayList<>();
        articleRepository.findAll().forEach(e -> list.add(e));
        return list;
    }

    public synchronized boolean addArticle(Article article){
            List<Article> list = articleRepository.findByTitleAndCategory(article.getTitle(), article.getCategory());
                if (list.size() > 0) {
                   return false;
                } else {
                articleRepository.save(article);
                return true;
       }
    }

    public void updateArticle(Article article) {
        articleRepository.save(article);
    }

    public void deleteArticle(int articleId) {
        articleRepository.delete(getArticleById(articleId));
    }
}
```

Inyectando el repositorio

usando el repositorio

Más custom Queries

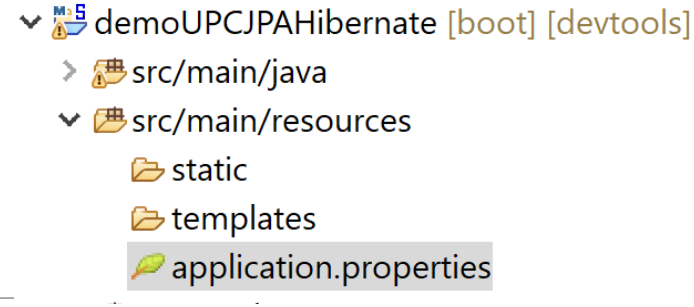
Dentro del repositorio:

```
@Query("SELECT f FROM Foo f WHERE LOWER(f.name) = LOWER(:name)")
Foo retrieveByName(@Param("name") String name);
```

Completo:

```
public interface ProductoRepositorio extends CrudRepository<Producto, Long>{
     @Query("SELECT a FROM Producto a WHERE a.codigo=:codigo and a.precio=:precio")
      List<Producto> fetchProductos(@Param("codigo") Long codigo, @Param("precio") double precio);
     @Query(value = "select codigo, descripcion ,precio from PRODUCTO_TP where codigo = ?1", nativeQuery = true)
      Producto findProductNative(Long codigo);
}
```

application.properties




```
1  ## Spring DATASOURCE (DataSourceAutoConfiguration & DataSourceProperties)
2  spring.datasource.url = jdbc:mysql://localhost:3306/bank?useSSL=false
3  spring.datasource.username = root
4  spring.datasource.password = root
5
6
7  ## Hibernate Properties
8  # The SQL dialect makes Hibernate generate better SQL for the chosen database
9  spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL5InnoDBDialect
10
11 # Hibernate ddl auto (create, create-drop, validate, update)
12 spring.jpa.hibernate.ddl-auto = update
```