FORMADORES { IT }

Curso Spring Boot

Temario

- **Spring JDBC/JPA**
  - ¿Qué es?
  - Responsabilidades Spring/Desarrollador
  - JDBC Template
  - Repositorios
  - Transacciones

JDBC

- ¿Qué es?
  - Es un módulo de Spring framework, más concretamente del paquete Spring Data el cual está dedicado específicamente para la conexión y operaciones con una Base de Datos
  - Permite implementar de una forma fácil los repositorios basados en JDBC
  - Al mantenerse sencillo en conceptos, Spring Data JDBC no ofrece características similares a JPA como, caching o lazy loading
  - Principios del diseño Spring Data JDBC
    - Si cargas una entidad, se ejecutan las sentencias SQL. Una vez hecho esto, tienes una entidad completamente cargada. No se realiza ninguna carga diferida o almacenamiento en caché.
    - Si guarda una entidad, se guarda. Si no es así, no es así. No hay seguimiento sucio ni sesión.
    - Existe un modelo simple de cómo asignar entidades a tablas. Probablemente solo funcione para casos bastante simples. Permite codificar una estrategia propia. Spring Data JDBC ofrece solo un soporte muy limitado para personalizar la estrategia con anotaciones.

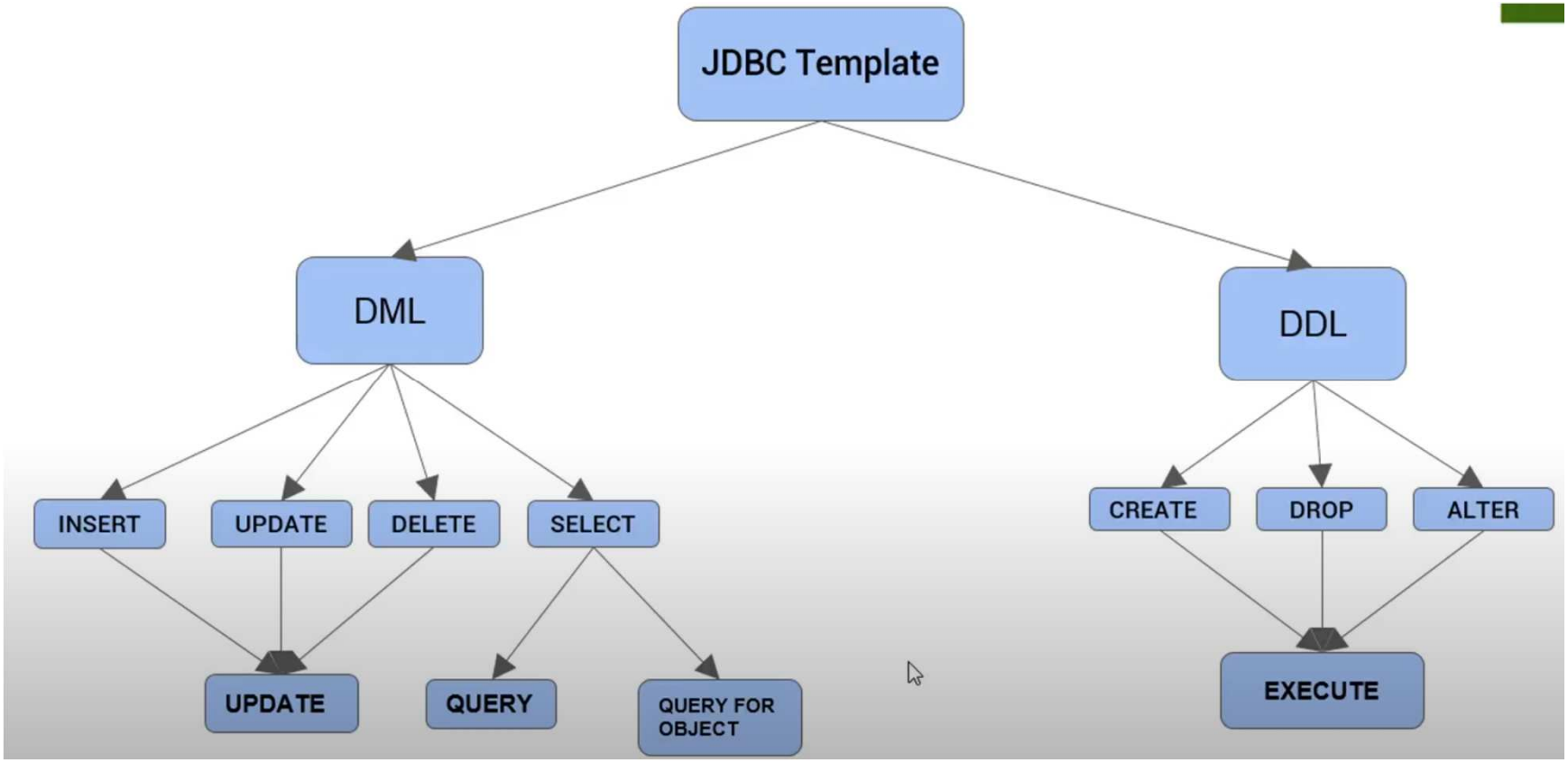
JDBC Template

- ¿Qué es?
  - Es un potente mecanismo para conectar a la Base de Datos y ejecutar consultas SQL. Internamente utiliza la API JDBC pero elimina bastantes problemas de la misma
- Problemas de JDBC API
  - Necesitamos escribir mucho código antes y después de ejecutar la consulta, como crear una conexión, una declaración, cerrar un conjunto de resultados, una conexión, etc.
  - Necesitamos realizar un código de manejo de excepciones en la lógica de la base de datos.
  - Necesitamos manejar la transacción.
  - La repetición de todos estos códigos de una lógica de base de datos a otra es una tarea que requiere mucho tiempo.
- Clases:
  - JdbcTemplate, NamedParameterJdbcTemplate, SimpleJdbcTemplate, SimpleJdbcInsert y SimpleJdbcCall

¿Quién hace qué?

Acción	Spring	Desarrollador
Definir los parámetros de conexión		X
Abrir la conexión	X	
Especificar el SQL statement		X
Declarar parámetros y proveer los valores a dichos parámetros		X
Preparar y ejecutar el statement	X	
Preparar el loop para iterar sobre los resultados	X	
Realizar el trabajo por cada iteración		X
Procesar excepciones	X	
Manejar transacciones	X	
Cerrar conexión, statement y resulset	X	

JDBC Template



JDBC Template

- Query básica

```
int result = jdbcTemplate.queryForObject(
    "SELECT COUNT(*) FROM EMPLOYEE", Integer.class);
```

- INSERT

```
public int addEmplyee(int id) {
    return jdbcTemplate.update(
      "INSERT INTO EMPLOYEE VALUES (?, ?, ?, ?)", id, "Bill", "Gates", "USA");
}
```

- Named Parameters (NamedParameterJdbcTemplate)

```
SqlParameterSource namedParameters = new MapSqlParameterSource().addValue("id", 1);
return namedParameterJdbcTemplate.queryForObject(
  "SELECT FIRST_NAME FROM EMPLOYEE WHERE ID = :id", namedParameters, String.class);
```

```
Employee employee = new Employee();
employee.setFirstName("James");

String SELECT_BY_ID = "SELECT COUNT(*) FROM EMPLOYEE WHERE FIRST_NAME = :firstName";

SqlParameterSource namedParameters = new BeanPropertySqlParameterSource(employee);
return namedParameterJdbcTemplate.queryForObject(
  SELECT_BY_ID, namedParameters, Integer.class);
```

JDBC Template

- Mapeo de resultados a Objetos Java

```
public class EmployeeRowMapper implements RowMapper<Employee> {
    @Override
    public Employee mapRow(ResultSet rs, int rowNum) throws SQLException {
        Employee employee = new Employee();

        employee.setId(rs.getInt("ID"));
        employee.setFirstName(rs.getString("FIRST_NAME"));
        employee.setLastName(rs.getString("LAST_NAME"));
        employee.setAddress(rs.getString("ADDRESS"));

        return employee;
    }
}
```

```
String query = "SELECT * FROM EMPLOYEE WHERE ID = ?";
Employee employee = jdbcTemplate.queryForObject(
  query, new Object[] { id }, new EmployeeRowMapper());
```

JDBC Template

- Procedimientos almacenados (SimpleJdbcCall)

```
SimpleJdbcCall simpleJdbcCall = new SimpleJdbcCall(dataSource)
                                    .withProcedureName("READ_EMPLOYEE");
```

```
public Employee getEmployeeUsingSimpleJdbcCall(int id) {
    SqlParameterSource in = new MapSqlParameterSource().addValue("in_id", id);
    Map<String, Object> out = simpleJdbcCall.execute(in);

    Employee emp = new Employee();
    emp.setFirstName((String) out.get("FIRST_NAME"));
    emp.setLastName((String) out.get("LAST_NAME"));

    return emp;
}
```

JDBC Template

- Operaciones Batch

```
public int[] batchUpdateUsingJdbcTemplate(List<Employee> employees) {
    return jdbcTemplate.batchUpdate("INSERT INTO EMPLOYEE VALUES (?, ?, ?, ?)",
        new BatchPreparedStatementSetter() {
            @Override
            public void setValues(PreparedStatement ps, int i) throws SQLException {
                ps.setInt(1, employees.get(i).getId());
                ps.setString(2, employees.get(i).getFirstName());
                ps.setString(3, employees.get(i).getLastName());
                ps.setString(4, employees.get(i).getAddress();
            }
            @Override
            public int getBatchSize() {
                return 50;
            }
        });
}
```

Repositorios

- ¿Qué son?
  - Media entre las capas de dominio y mapeo de datos utilizando una interfaz a modo de colección para acceder a los objetos de dominio
- Tipos de Repository
  - **CrudRepository:** Implementa las operaciones más comunes save, delete, findById…

```
@Repository
public interface PersonRepository extends CrudRepository<Person, Long> {
}
```

  - **PagingAndSortingRepository:** Implementa operaciones de paginación y ordenado. Uso común para las interfaces REST

```
@NoRepositoryBean
public interface PagingAndSortingRepository<T, ID> extends CrudRepository<T, ID> {
    Iterable<T> findAll(Sort var1);

    Page<T> findAll(Pageable var1);
}
```

Repositorios

- Métodos derivados
  - Filtro por nombre de campo: List<User> **findBy**Name(String name)
  - **Find, read, query, count y get**: List<User> **queryBy**Name(String name)
  - Eliminar duplicados con **Distinct, First y Top**: List<User> find**Top3**ByAge()
  - Condiciones de igualdad Is y Equals: List<User> **findBy**Name**Is**(String name)
  - Negación **Not**: List<User> **findBy**Name**IsNot**(String name)
  - Nullable: List<User> **findBy**Name**IsNull**()
  - Verdadero o false: List<User> **findBy**Active**False**()
  - Condiciones: **StartingWith, EndingWith, Containing, Like**: findByName**StartingWith** (String prefix)
  - Comparaciones: LessThan, LessThanEqual, GreaterThan, GreaterThanEqual, Between, In, After, Before: List<User> findByAge**In**(Collection<Integer> ages);
  - Ordenación: OrderBy..ASC/DESC: findByName**OrderBy**Name**Desc**(String name)

Transaciones

- @Transactional
  - Anotación para establecer la configuración de inicio/fin de la transacción, propagación
- Propagación
  - **REQUIRED**: Da soporte a la transacción actual o crea una nueva sino existe.
  - **SUPPORTS**: Da soporte a la transacción actual, sino existe, ejecuta el método sin transacción.
  - **MANDATORY**: Da soporte a la transacción actual, sino existe, lanza una excepción.
  - **REQUIRES_NEW**: Crea una nueva transacción, si existe una, la suspende.
  - **NOT_SUPPORTED**: Ejecuta el método sin transacción, sí existe, la suspende.
  - **NEVER:** Ejecuta el método sin transacción, sí existe, lanza una excepción.
  - **NESTED**: Ejecuta dentro una transacción si existe una.

Transaciones

- Aislamiento
  - El nivel de aislamiento se refiere al grado de aislamiento entre varias transacciones concurrentes.
  - ISOLATION_DEFAULT: este es el valor predeterminado, lo que significa que se utiliza el nivel de aislamiento predeterminado de la base de datos subyacente. Para la mayoría de las bases de datos, este valor suele ser ISOLATION_READ_COMMITTED.
  - ISOLATION_READ_UNCOMMITTED: este nivel de aislamiento indica que una transacción puede leer datos modificados por otra transacción pero aún no ha confirmado datos. Este nivel no puede evitar lecturas sucias, lecturas no repetibles y lecturas mágicas, por lo que este nivel de aislamiento rara vez se usa. Por ejemplo, PostgreSQL en realidad no tiene este nivel.
  - ISOLATION_READ_COMMITTED: este nivel de aislamiento significa que una transacción solo puede leer datos confirmados por otra transacción. Este nivel evita lecturas sucias, que es el valor recomendado en la mayoría de los casos.
  - ISOLATION_REPEATABLE_READ: este nivel de aislamiento significa que una transacción puede ejecutar repetidamente una consulta varias veces durante todo el proceso, y los registros devueltos cada vez son los mismos. Este nivel evita lecturas sucias y lecturas no repetibles.
  - ISOLATION_SERIALIZABLE: todas las transacciones se ejecutan una tras otra, por lo que no hay posibilidad de interferencia entre transacciones, es decir, este nivel puede evitar lecturas sucias, lecturas no repetibles y lecturas mágicas. Pero esto afectará seriamente el desempeño del programa. Normalmente, este nivel no se usa.

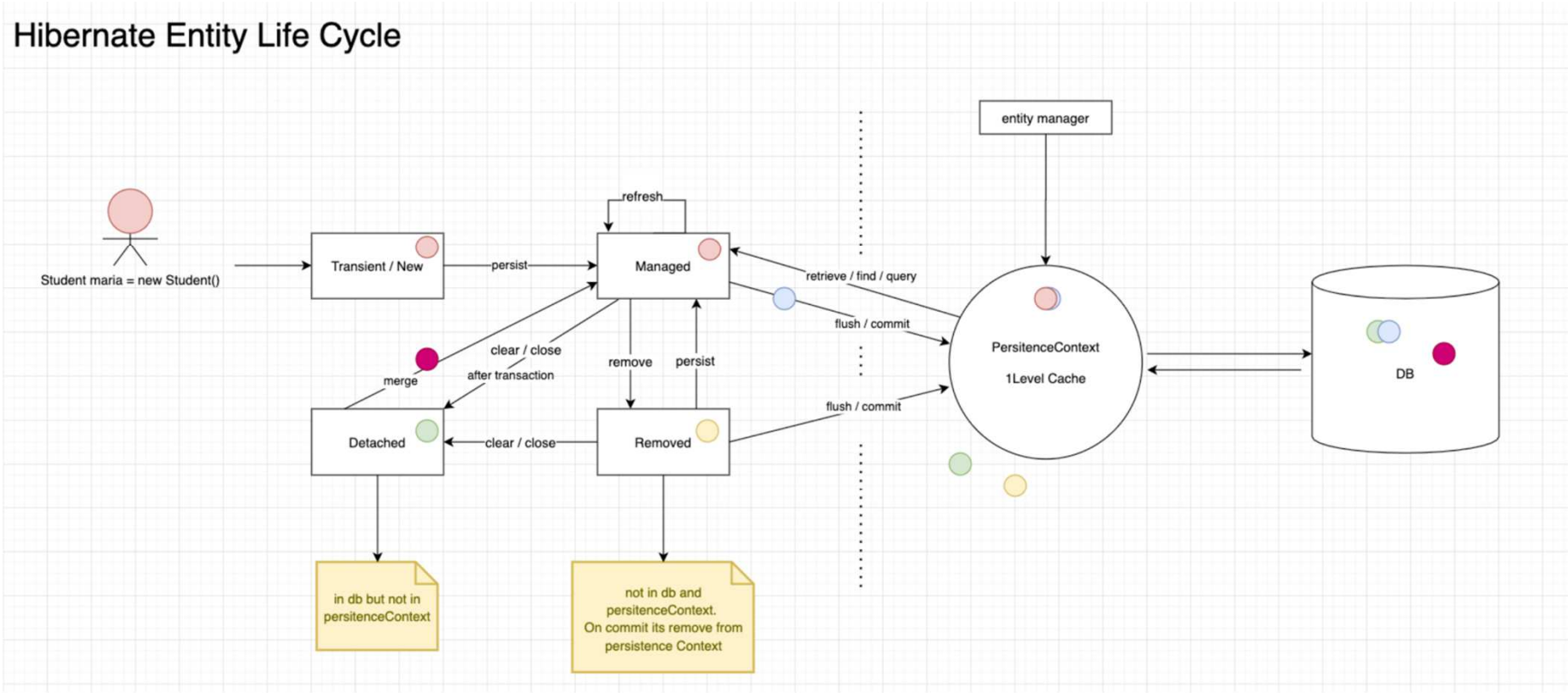
FORMADORES { IT }

@Transactional

Atributos	Tipos de	descripción
value	String	Descriptor calificado opcional, especifica el administrador de transacciones a utilizar
propagation	enum: Propagation	Configuraciones opcionales de comportamiento de propagación de transacciones
isolation	enum: Isolation	Configuración opcional del nivel de aislamiento de transacciones
readOnly	boolean	Lectura y escritura o transacciones de solo lectura, la lectura y escritura predeterminada
timeout	int (in seconds granularity)	Configuración de tiempo de espera de transacción
rollbackFor	Matriz de objetos de clase	Debe heredar de la matriz de clase Throwable Exception que hace que la transacción retroceda
rollbackForClassName	Matriz de nombre de clase	Debe heredar de Throwable Array los nombres de clase de excepción que hacen que la transacción se revierta
noRollbackFor	Matriz de objetos de clase	Debe heredar de una matriz de clases de excepción que Throwable no hará que la transacción se revierta
noRollbackForClassName	Matriz de nombre de clase	Debe heredar de Throwable Array los nombres de clase de excepción que no harán que la transacción se revierta

FORMADORES { IT }

JPA



FORMADORES { IT }

Schema