Spring Boot
JDBC database access

springboot05

Spring Boot JDBC

- **Spring Boot JDBC** fornisce starter e librerie per connettere un'applicazione con JDBC semplificando l'utilizzo di tutte le funzioni
- JDBC è l'interfaccia base, meno astratta, di accesso e gestione al database e tutte le tecnologie, come Hibernate, la utilizzano tecnicamente
- In Spring Boot JDBC i beans di gestione del database sono:
  - **DataSource**
  - **JdbcTemplate**
  - **NamedParameterJdbcTemplate**

```
@Autowired
JdbcTemplate jdbcTemplate;
@Autowired
private NamedParameterJdbcTemplate jdbcTemplate;
```

Spring Boot JDBC

- **Spring Boot JDBC** elimina la gestione diretta di:
  - Creazione e chiusura di *connessoni*
  - Creazione ed esecuzione di *statements* e *stored procedure*
  - Iterazione sui *ResultSet* e restituzione dei risultati
- **JdbcTemplate**
  - I parametri della query sono posizionali

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

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

Spring Boot JDBC Query con namedParameter

- **NamedParameterJdbcTemplate**
  - I parametri della query sono identificati da un nome
  - Si fa riferimento all'oggetto (bean) da mappare con i dati letti
  - I parametri devono avere lo stesso nome degli attributi del bean

```
23      @Override
24      public int update(Book book) {
25          return namedParameterJdbcTemplate.update(
26                  "update books set price = :price where id = :id",
27                  new BeanPropertySqlParameterSource(book));
28      }
29
```

Spring Boot JDBC Query con namedParameter 1

- **NamedParameterJdbcTemplate**
  - I parametri della query sono identificati da un nome
  - Si fa riferimento a una Map con i parametri key/value
  - ***queryForObject*** restituisce un solo oggetto o nessun oggetto

```
30      @Override
31      public Optional<Book> findById(Long id) {
32          return namedParameterJdbcTemplate.queryForObject(
33                  "select * from books where id = :id",
34                  new MapSqlParameterSource("id", id),
35                  (rs, rowNum) ->
36                          Optional.of(new Book(
37                                  rs.getLong("id"),
38                                  rs.getString("name"),
39                                  rs.getBigDecimal("price")
40                          ))
41          );
42      }
```

Spring Boot JDBC Query con namedParameter 2

- **NamedParameterJdbcTemplate**
  - I parametri della query sono identificati da un nome
  - Si fa riferimento a una Map con i parametri key/value
  - ***query*** restituisce una Collection di oggetti

```
44    @Override
45    public List<Book> findByNameAndPrice(String name, BigDecimal price) {
46
47        MapSqlParameterSource mapSqlParameterSource = new MapSqlParameterSource();
48        mapSqlParameterSource.addValue("name", "%" + name + "%");
49        mapSqlParameterSource.addValue("price", price);
50
51        return namedParameterJdbcTemplate.query(
52                "select * from books where name like :name and price <= :price",
53                mapSqlParameterSource,
54                (rs, rowNum) ->
55                        new Book(
56                                rs.getLong("id"),
57                                rs.getString("name"),
58                                rs.getBigDecimal("price")
59                        )
60        );
61    }
```

Spring Boot JDBC con namedParameter 3

- **RowMapper**
  - Effettua il mapping fra il ***ResultSet*** e l'oggetto da popolare
  - Può essere definito e allocato **esplicitamente**

```
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());
```

Spring Boot JDBC con namedParameter 4

- **RowMapper**
  - Effettua il mapping fra il ***ResultSet*** e l'oggetto da popolare
  - Può essere dichiarato **implicitamente** con notazione **Lambda**

```
30      @Override
31      public Optional<Book> findById(Long id) {
32          return namedParameterJdbcTemplate.queryForObject(
33                  "select * from books where id = :id",
34                  new MapSqlParameterSource("id", id),
35                      (rs, rowNum) ->
36                              Optional.of(new Book(
37                                      rs.getLong("id"),
38                                      rs.getString("name"),
39                                      rs.getBigDecimal("price")
40                              ))
41          );
42      }
43
```

Spring Boot JDBC plain

- Utilizzando **JDBC** in modo **nativo**, riferendosi al solo **DataSource**, si deve gestire *connessione*, *prepared statement* e *gestione errori*:

```
public Optional<Book> findByIdPlainJdbc(Long id) {
    String query = "select * from books where id = ?";
    Book book = null;
    Connection con = null;
    PreparedStatement ps = null;
    ResultSet rs = null;
    try{
        con = dataSource.getConnection();
        ps = con.prepareStatement(query);
        ps.setLong(1, id);
        rs = ps.executeQuery();
        if(rs.next()){
            book = new Book();
            book.setId(id);
            book.setName(rs.getString("name"));
            book.setPrice(rs.getBigDecimal("price"));
            System.out.println("Book Found::"+book);
        }else{
            System.out.println("No Book found with id="+id);
        }
    }catch(SQLException e){
        e.printStackTrace();
    }finally{
        try {
            rs.close();
            ps.close();
            con.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
    return Optional.of(book);
}
```

Spring Boot JDBC Batch

- **Spring Boot JDBC** supporta le operazioni batch via batchUpdate

```
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;
            }
        });
}
```

Spring Boot JDBC Batch 2

- Oppure, più semplicemente, usando NamedParameterJdbcTemplate:

```
SqlParameterSource[] batch = SqlParameterSourceUtils.createBatch(employees.toArray());
int[] updateCounts = namedParameterJdbcTemplate.batchUpdate(
    "INSERT INTO EMPLOYEE VALUES (:id, :firstName, :lastName, :address)", batch);
return updateCounts;
```

Spring Boot JDBC Exception Handling

- Spring JDBC in caso di exception lancia sempre ***DataAccessException*** e traduce tutte le exception Sql di basso livello con questa exception o una delle sue subclass
- In questo modo con Spring JDBC non ci si preoccupa delle exception di persistenza di **basso livello**
- Il meccanismo di handling delle exception è **indipendente** dal tipo di dbms sottostante
- E' comunque possibile personalizzare le exception e intercettare eventuali error code specifici
- La personalizzazione avviene in due fasi
  - Estendendo **SQLErrorCodeSQLExceptionTranslator**
  - **Dichiarando** l'exception custom a **jdbcTemplate**

Spring Boot JDBC Exception Handling 2

```
public class CustomSQLErrorCodeTranslator extends SQLErrorCodeSQLExceptionTranslator {
    @Override
    protected DataAccessException
      customTranslate(String task, String sql, SQLException sqlException) {
        if (sqlException.getErrorCode() == 23505) {
          return new DuplicateKeyException(
            "Custom Exception translator - Integrity constraint violation.", sqlException);
        }
        return null;
    }
}
```

E, per utilizzare l'exception personalizzata:

```
CustomSQLErrorCodeTranslator customSQLErrorCodeTranslator =
  new CustomSQLErrorCodeTranslator();
jdbcTemplate.setExceptionTranslator(customSQLErrorCodeTranslator);
```

Spring Boot JDBC Configurazione

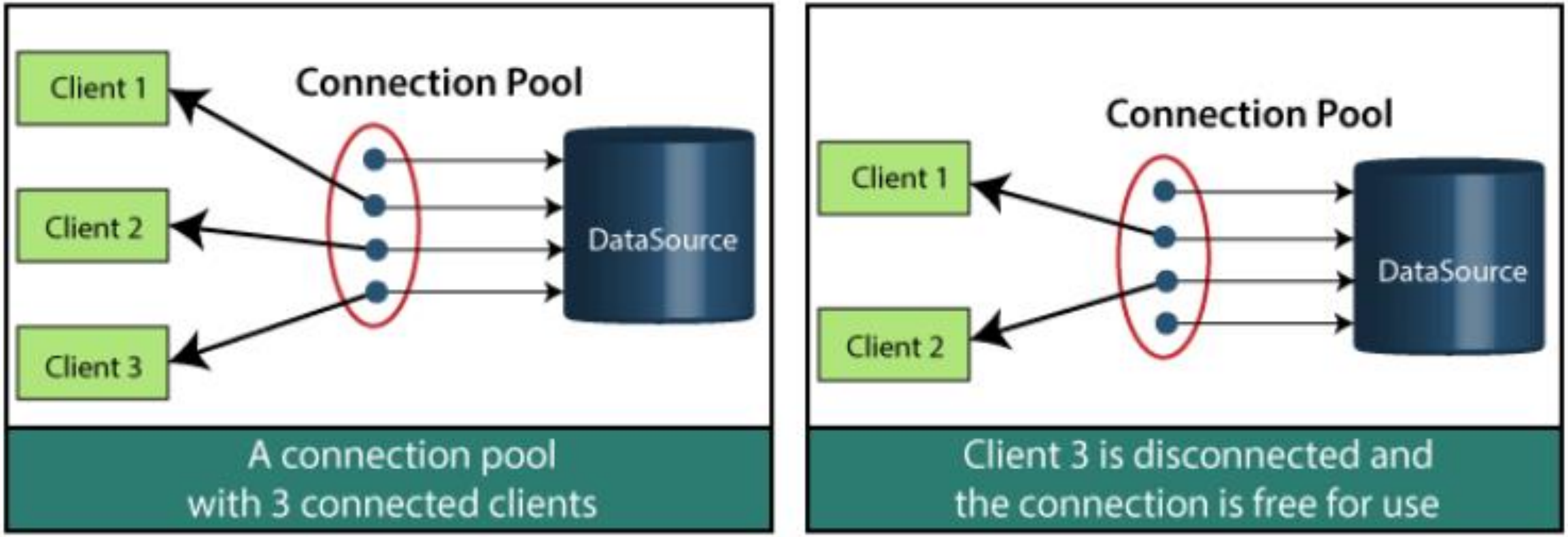
- Nel file **application.properties** viene configurato il **DataSource** e il **connection pooling**
- **JDBC connection pooling** è un meccanismo per gestire richieste di connessione **multiple** al database
- Le connessioni al database sono molto **costose** in termini di risorse e devono essere sempre riutilizzate, Spring Boot JDBC facilita il riuso delle connessioni al database attraverso una cache in memoria chiamata **connection pool**
- Il connection pool è uno strato software in cima a ogni driver standard JDBC

Spring Boot JDBC Configurazione

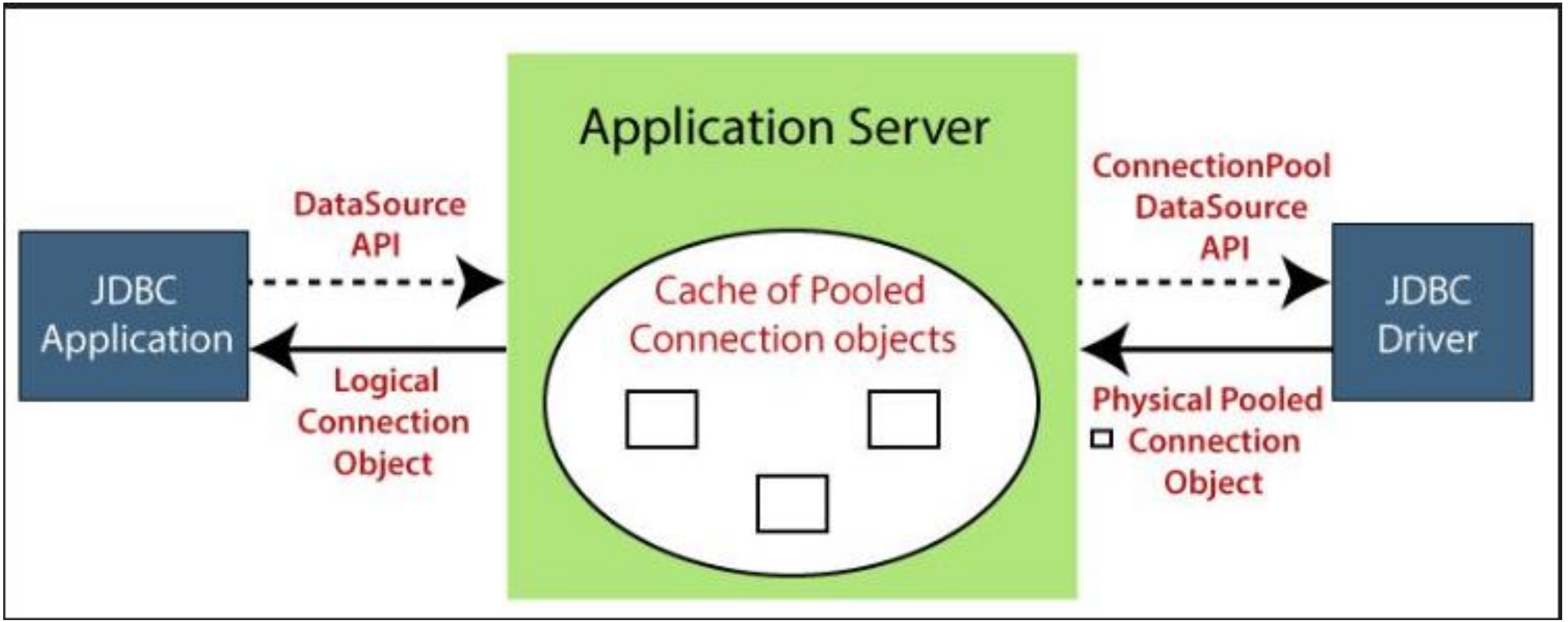
- Il connection pool ha le seguenti funzioni
  - Gestione connessioni disponibili
  - Allocare nuove connessioni
  - Chiudere connessioni
- Il connection pool di default in Spring Boot è **HikariCP**
- Altri connection pool utilizzabili e configurabili sono:
  - **Tomcat JDBC Connection Pool**
  - **Apache Commons DBCP2**

```
spring.datasource.url=jdbc:mysql://localhost:3306/SPFORMAZIONE
spring.datasource.username=GZEDDA
spring.datasource.password=giampietro4
spring.datasource.dbcp2.max-total=1
spring.datasource.hikari.connection-timeout=20000
spring.datasource.hikari.minimum-idle=5
spring.datasource.hikari.maximum-pool-size=12
spring.datasource.hikari.idle-timeout=300000
spring.datasource.hikari.max-lifetime=1200000
```

Spring Boot JDBC Schema



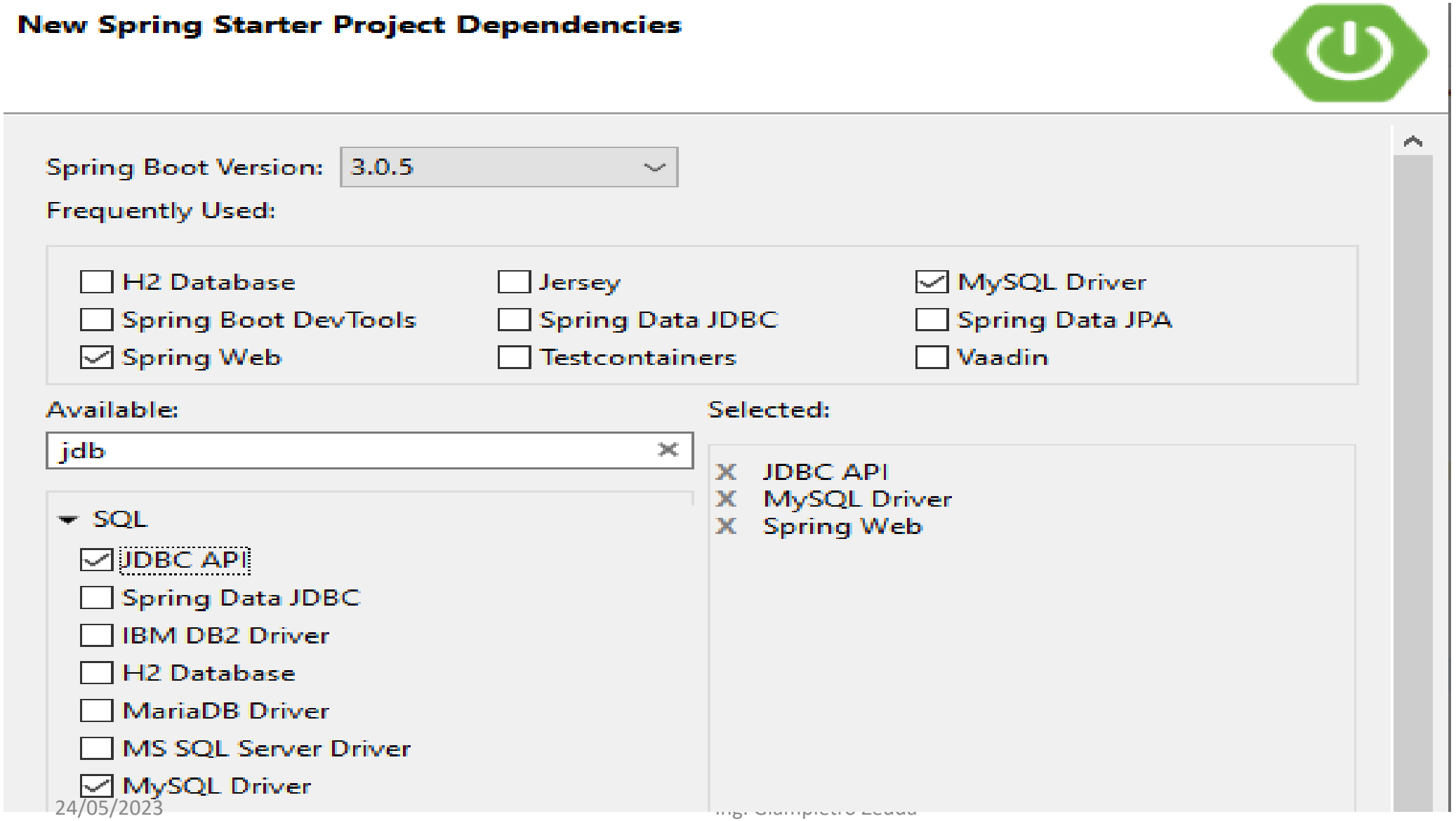
Spring Boot JDBC Caching



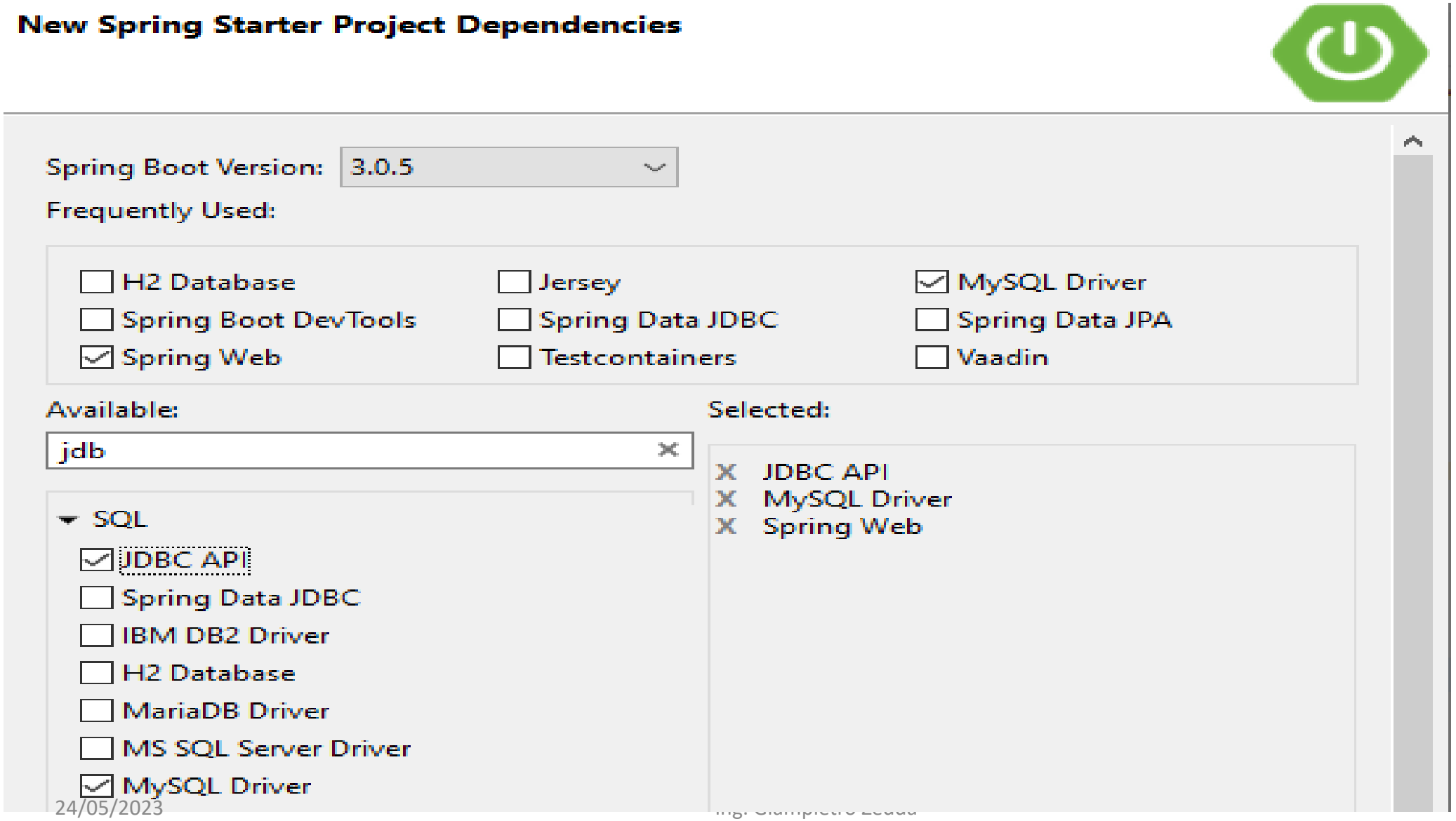
Spring Boot JDBC vs. Hibernate

DBC	Hibernate
JDBC is a **technology**.	Hibernate is an **ORM** framework.
In JDBC, the user is responsible for creating and closing the connections.	In Hibernate, the run time system takes care of creating and closing the connections.
It does not support lazy loading.	It supports lazy loading that offers better performance.
It does not support associations (the connection between two separate classes).	It supports associations

Spring Boot JDBC project



Spring Boot JDBC project



Spring Boot JDBC project

```
14 @SpringBootApplication
15 public class SpringBoot05RestJdbcMySqlApplication implements CommandLineRunner{
16     private static final Logger log = (Logger) LogManager.getLogger(SpringBoot05RestJdbcMySqlApplic
17
18     @Autowired
19     JdbcTemplate jdbcTemplate;
20
21     @Autowired
22     @Qualifier("jdbcBookRepository")                    // Test JdbcTemplate
23    //@Qualifier("namedParameterJdbcBookRepository")  // Test NamedParameterJdbcTemplate
24     private BookRepository bookRepository;
25
26     public static void main(String[] args) {
27         SpringApplication.run(SpringBoot05RestJdbcMySqlApplication.class, args);
28     }
29
30     @Override
31     public void run(String... args) throws Exception {
32         log.info("==== Creating tables for testing...");
33         jdbcTemplate.execute("DROP TABLE IF EXISTS books ");
34         jdbcTemplate.execute("CREATE TABLE books" +
35                 "(id SERIAL, name VARCHAR(255), price NUMERIC(15, 2))");
36         jdbcTemplate.execute("insert into books (name, price) values('Lo Zen', 20) ");
37         jdbcTemplate.execute("insert into books (name, price) values('Yoga', 30) ");
38     }
39 }
```

Spring Boot JDBC project Controller

```
22
23 @RestController
24 @RequestMapping("springboot05")
25 public class SpringBootJdbcController {
26
27     @Autowired
28     JdbcTemplate jdbc;
29
30     @Autowired
31     @Qualifier("namedParameterJdbcBookRepository")     // Test JdbcTemplate
32 //  @Qualifier("jdbcBookRepository")                   // Test JdbcTemplate
33     JdbcBookRepository repository;
34
35     @GetMapping(value = "/count")
36     public String count() {
37         return "Numero libri inseriti:"+repository.count();
38     }
39     @PostMapping(value = "/save")
40     public String save(@RequestBody Book book) {
41         return repository.save(book) > 0 ? "Libro Salvato" : "Libro Non Salvato";
42     }
43     @PutMapping("/update")
44     public String update(@RequestBody Book book) {
45         return "Aggiornati " + repository.update(book) + "Libri";
46     }
```

Spring Boot JDBC project Controller

```
47      @DeleteMapping(value = "/delete/{id}")
48      public String delete(@PathVariable("id") Long id) {
49          return "Deletati " + repository.deleteById(id) + " Libri";
50      }
51      @GetMapping(value = "/findAll")
52      public List<Book> findAll() {
53          return repository.findAll();
54      }
55
56      @GetMapping(value = "/findByNameAndPrice/{name}/{price}")
57      public List<Book> findByNameAndPrice(@PathVariable("name") String name
58                                           , @PathVariable("price") BigDecimal price) {
59          return repository.findByNameAndPrice(name, price);
60      }
61
62      @GetMapping(value = "/findById/{id}")
63      public Optional<Book> findById(@PathVariable("id") Long id) {
64          return repository.findById(id);
65      }
66      @GetMapping(value = "/getNameById/{id}")
67      public String getNameById(@PathVariable("id") Long id) {
68          return "Book name with id " + id + " is " + repository.getNameById(id);
69      }
70  }
```

Spring Boot JDBC project Model

```java
package com.demo.repository;

import java.math.BigDecimal;

public interface BookRepository {

    int count();

    int save(Book book);

    int update(Book book);

    int deleteById(Long id);

    List<Book> findAll();

    List<Book> findByNameAndPrice(String name, BigDecimal price);

    Optional<Book> findById(Long id);

    String getNameById(Long id);

}
```

```java
package com.demo.entity;

import java.math.BigDecimal;

public class Book {

    private Long id;
    private String name;
    private BigDecimal price;

    public Book(long id, String name, BigDecimal price) {
        this.id = id;
        this.name = name;
        this.price = price;
    }

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }
```

Spring Boot JDBC JdbcBookRepository

```java
@Repository
@Component
public class JdbcBookRepository implements BookRepository {

    // Spring Boot will create and configure DataSource and JdbcTemplate
    @Autowired
    private JdbcTemplate jdbcTemplate;

    @Override
    public int count() {
        return jdbcTemplate
                .queryForObject("select count(*) from books", Integer.class);
    }

    @Override
    public int save(Book book) {
        return jdbcTemplate.update(
                "insert into books (name, price) values(?,?)",
                book.getName(), book.getPrice());
    }

    @Override
    public int update(Book book) {
        return jdbcTemplate.update(
                "update books set price = ? where id = ?",
                book.getPrice(), book.getId());
    }
```

```java
    @Override
    public int deleteById(Long id) {
        return jdbcTemplate.update(
                "delete from books where id = ?",
                id);
    }

    @Override
    public List<Book> findAll() {
        return jdbcTemplate.query(
                "select * from books",
                (rs, rowNum) ->
                        new Book(
                                rs.getLong("id"),
                                rs.getString("name"),
                                rs.getBigDecimal("price")
                        )
        );
    }
```

Spring Boot JDBC JdbcBookRepository

```
61      // jdbcTemplate.queryForObject, populates a single obj
62      @SuppressWarnings("deprecation")
63      @Override
64      public Optional<Book> findById(Long id) {
65          return jdbcTemplate.~~queryForObject~~(
66                  "select * from books where id = ?",
67                  new Object[]{id},
68                  (rs, rowNum) ->
69                          Optional.of(new Book(
70                                  rs.getLong("id"),
71                                  rs.getString("name"),
72                                  rs.getBigDecimal("price")
73                          ))
74          );
75      }
```

```
77      @SuppressWarnings("deprecation")
78      @Override
79      public List<Book> findByNameAndPrice(String name, BigDecimal price) {
80          return jdbcTemplate.~~query~~(
81                  "select * from books where name like ? and price <= ?",
82                  new Object[]{"%" + name + "%", price},
83                  (rs, rowNum) ->
84                          new Book(
85                                  rs.getLong("id"),
86                                  rs.getString("name"),
87                                  rs.getBigDecimal("price")
88                          )
89          );
90      }
```

```
92      @SuppressWarnings("deprecation")
93      @Override
94      public String getNameById(Long id) {
95          return jdbcTemplate.~~queryForObject~~(
96                  "select name from books where id = ?",
97                  new Object[]{id},
98                  String.class
99          );
100     }
101
102 }
```

Spring Boot JDBC NamedParameterJdbcBookRepository

```java
@Repository
@Component
public class NamedParameterJdbcBookRepository extends JdbcBookRepository {

    @Autowired
    private NamedParameterJdbcTemplate namedParameterJdbcTemplate;

    @Override
    public int update(Book book) {
        return namedParameterJdbcTemplate.update(
                "update books set price = :price where id = :id",
                new BeanPropertySqlParameterSource(book));
    }

    @Override
    public Optional<Book> findById(Long id) {
        return namedParameterJdbcTemplate.queryForObject(
                "select * from books where id = :id",
                new MapSqlParameterSource("id", id),
                (rs, rowNum) ->
                        Optional.of(new Book(
                                rs.getLong("id"),
                                rs.getString("name"),
                                rs.getBigDecimal("price")
                        ))
        );
    }
```