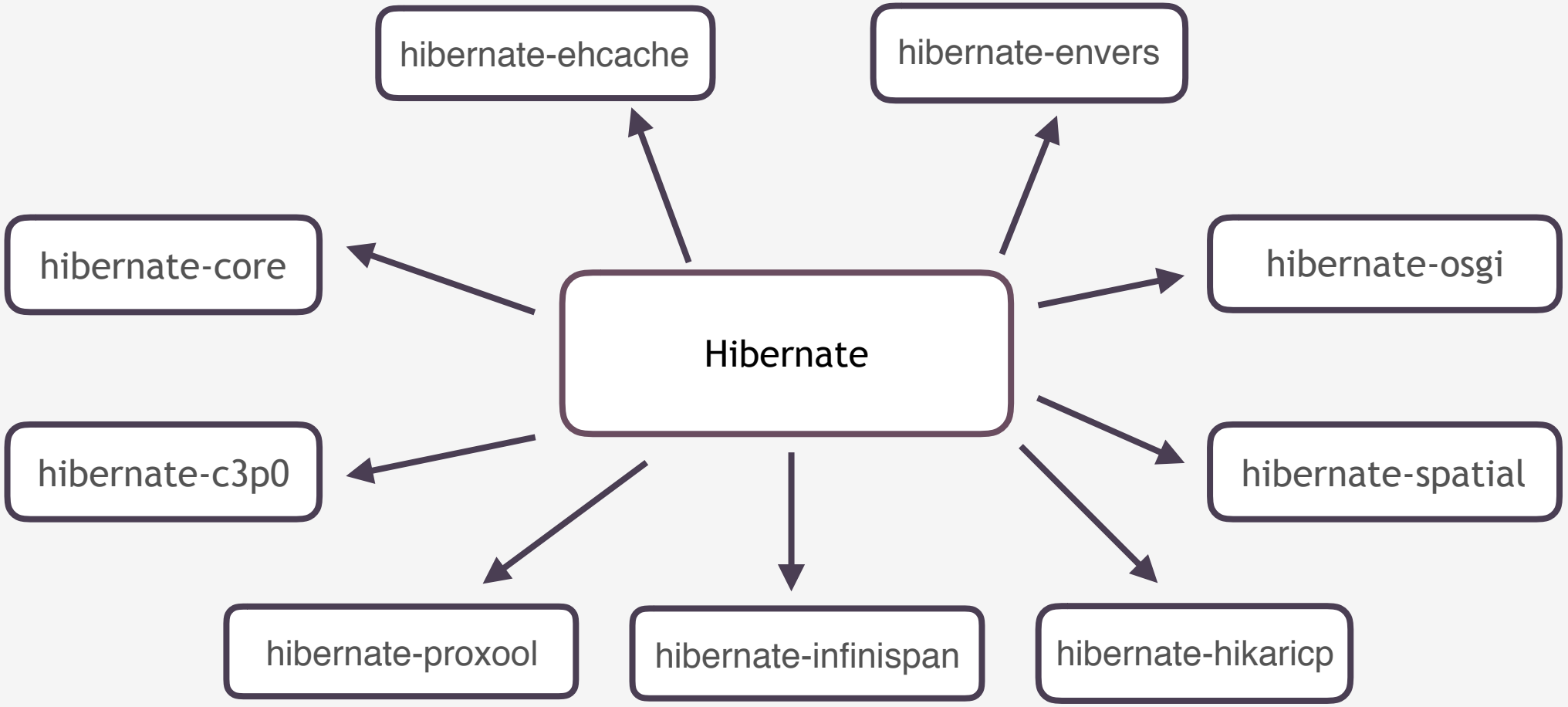
Hibernate

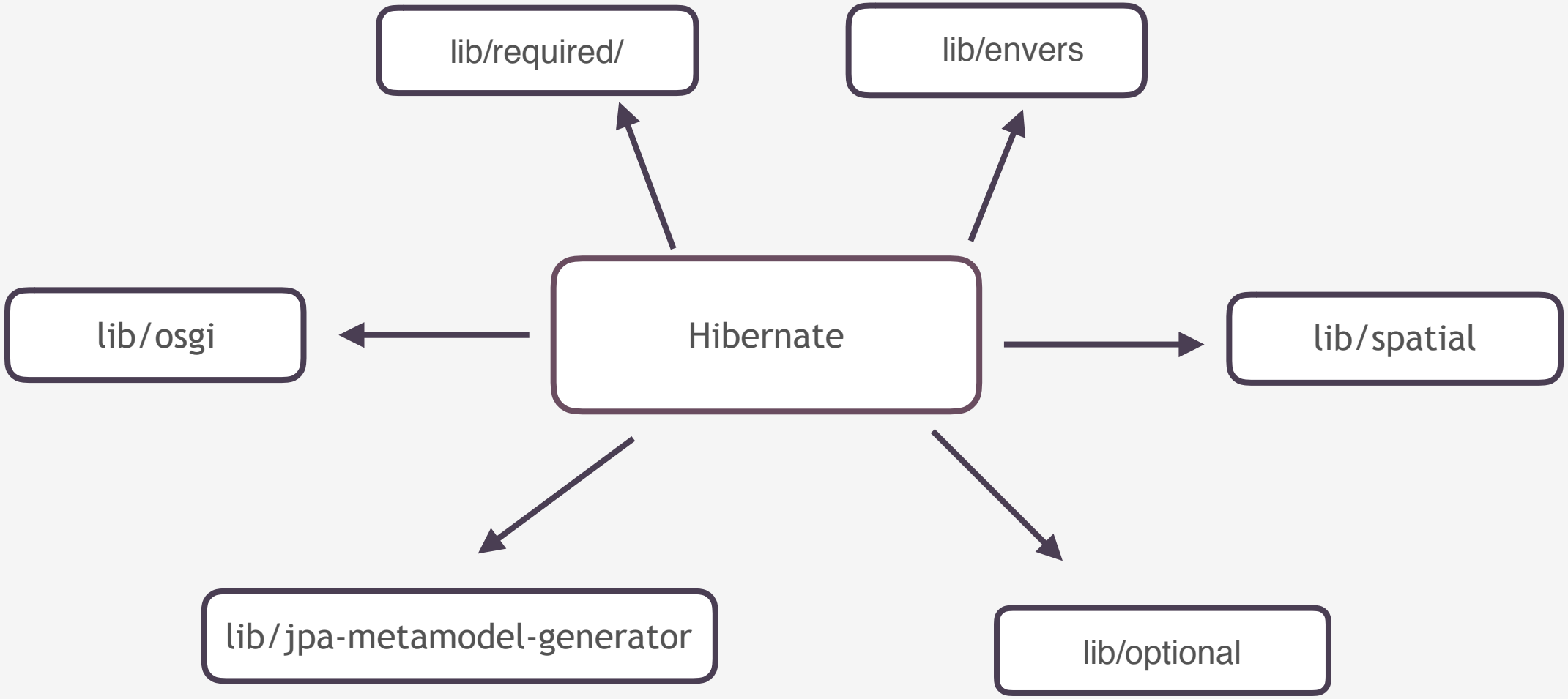
Piotr Brzozowski
SDA

Hibernate - Podstawy

Moduły Hibernate



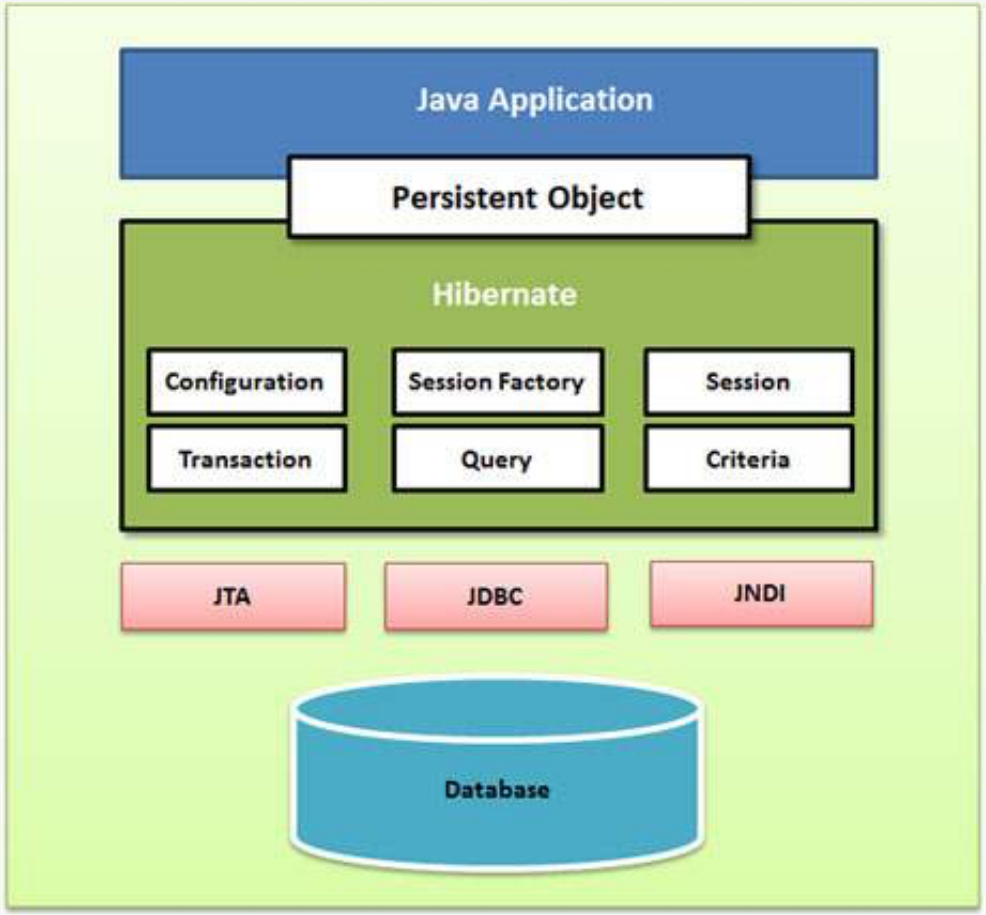
Pakiety Hibernate



Architektura JDBC

Główne elementy:

- ORM (Object-Relational Mapping)
- Connection Manager
- Transaction Manager
- HQL



Zalety i wady Hibernate

Zalety

- wsparcie dla HQL
- automatyczne mapowanie obiektów Javy z tabelami bazy danych
- rozwiązanie typu ORM
- łatwa optymalizacja
- koszt utrzymania
- rozwiązanie typu open-source
- skalowalność rozwiązania
- automatyczne wersjonowanie

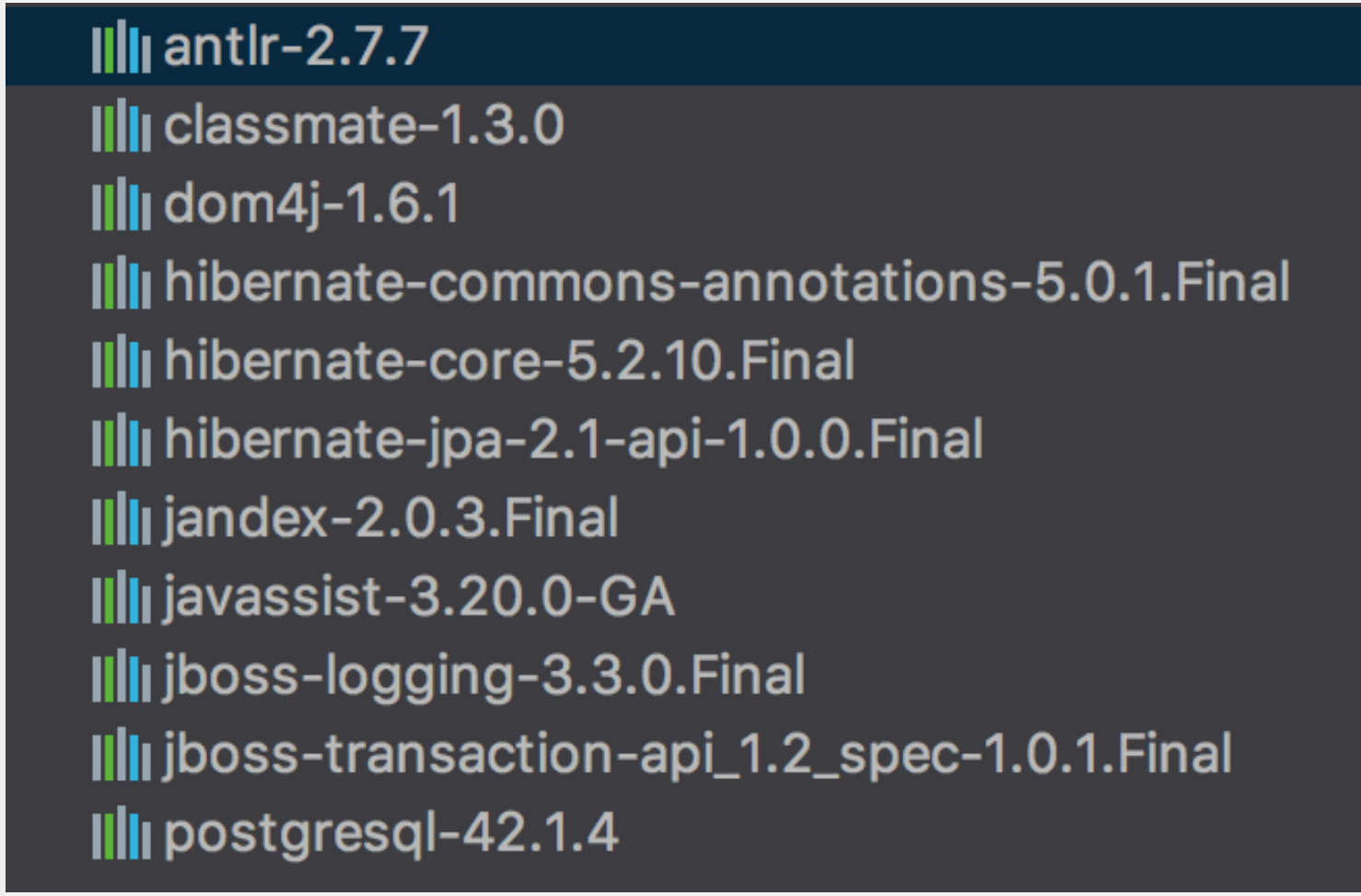
Wady

- złożone dane zmniejszają wydajność
- duże zróżnicowanie i złożoność aplikacji
- dodatkowa warstwa abstrakcji przy niektórych projektach zwiększa złożoność
- utrzymanie projektu na bazie Hibernate wymaga jego znajomości przez developerów

Jak zacząć?

- Zaimportuje w projekcie poniższe pliki jar z pakietu startowego Hibernate

  http://hibernate.org/orm/downloads/



Jak zacząć?

- Dodaj przedstawione zależności do pliku pom.xml
- sterownik bazy danych zależny od wykorzystanej bazy danych

```
<dependencies>
    <dependency>
        <groupId>org.hibernate</groupId>
        <artifactId>hibernate-core</artifactId>
        <version>5.2.17.Final</version>
    </dependency>
    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>5.1.6</version>
    </dependency>
</dependencies>
```

Jak zacząć?

- Stwórz plik hibernate.cfg.xml

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

    <session-factory>
        <property name="dialect">SQL for the chosen database.</property>
        <property name="connection.url">JDBC URL to the database instance</property>
        <property name="connection.username">username</property>
        <property name="connection.password">password</property>
        <property name="connection.driver_class">The JDBC driver class</property>
        <property name="“show_sql”">true/false - SQL app logs</property>

        <!— mapping —>
    </session-factory>

</hibernate-configuration>
```

Jak zacząć?

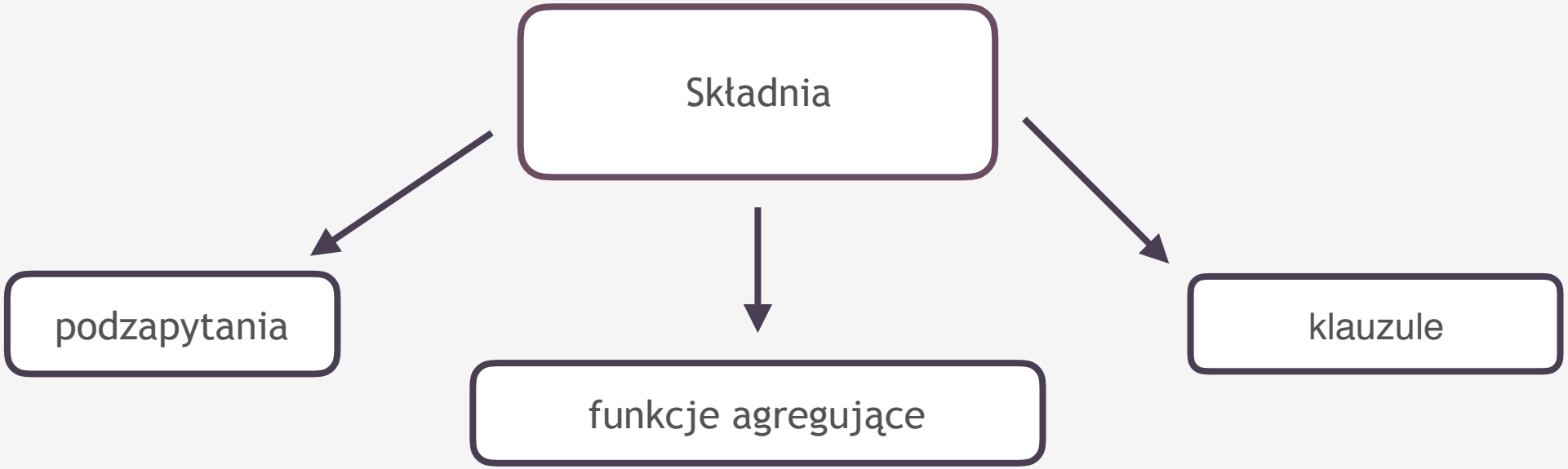
- Otwórz sesję za pomocą SessionFactory

```
private static final SessionFactory sessionFactory;
static {
    try {
        sessionFactory = new Configuration().configure("hibernate.cfg.xml")
        .buildSessionFactory();
    } catch (Throwable ex) {
        // Log exception!
        throw new ExceptionInInitializerError(ex);
    }
}

public static Session getSession()
        throws HibernateException {
    return sessionFactory.openSession();
}
```

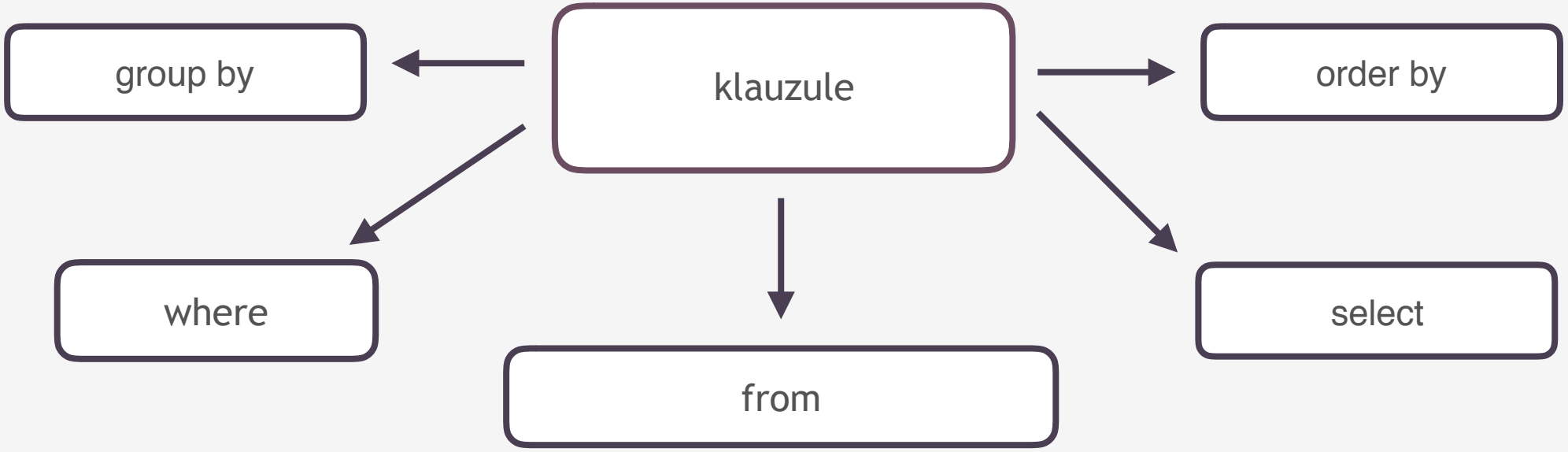
HQL

HQL - Hibernate Query Language

Hibernate Query Language jest językiem zapytań używanym na potrzeby Hibernate. HQL jest bardzo podobny do SQL, pozwala jednak na wykorzystanie polimorfizmu, asocjacji i innych cech modelu obiektowego w zapytaniach. Zapytania HQL są automatycznie przekładane na SQL, zaś wyniki zapytania są zwracane w formie referencji do odpowiednich obiektów.



HQL - klauzule



HQL - klauzula from

Klauzula **FROM** używana jest do wskazania, obiekty jakiej klasy chcemy odpytywać. Składnia klauzuli FROM wygląda następująco:

from **nazwa_klasy** [[as] alias]

```
Session session = sessionFactory.openSession();
Transaction tx = null;
try{
    tx = session.beginTransaction();
    List authors = session.createQuery("FROM Book").list();
    for (Iterator iterator =
         authors.iterator(); iterator.hasNext();){
        Book book = (Book) iterator.next();
        System.out.println(" Id: " + book.getId());
        System.out.println(" Book name: " + book.getName());
        System.out.println("  Book type: " + book.getBookType());
        System.out.println("  author: " + book.getAuthor().toString());
    }
    tx.commit();
}catch (HibernateException e) {
    if (tx!=null) tx.rollback();
    e.printStackTrace();
}finally {
    session.close();
}
```

HQL - klauzula select

Klauzula **SELECT** pozwala na określenie, które z atrybutów obiektu powinny zostać zwrócone przez zapytanie. Np:

```
select osoba.malzonek from Osoba osoba
```

```
String hql = "SELECT E.firstName FROM Employee E";
Query query = session.createQuery(hql);
List results = query.list();
```

HQL - klauzula where

Klauzula **WHERE** pozwala określić warunki, jakie powinny spełniać elementy wyniku zapytania. Jej składnia, znaczenie i możliwe do użycia wyrażenia są praktycznie identyczne ze znanymi z języka SQL, np:

```
from Osoba as osoba where osoba.imie='Fritz'
```

HQL - klauzula order by

Wynik zwracany przez zapytanie może być uporządkowany w.g. dowolnego atrybutu odpytywanej klasy lub jej składowej:

```
from Osoba osoba order by osoba.imie asc, osoba.wzrost desc, osoba.data_ur
```

```
String hql = "FROM Employee E WHERE E.id > 10 ORDER BY E.salary DESC";
Query query = session.createQuery(hql);
List results = query.list();
```

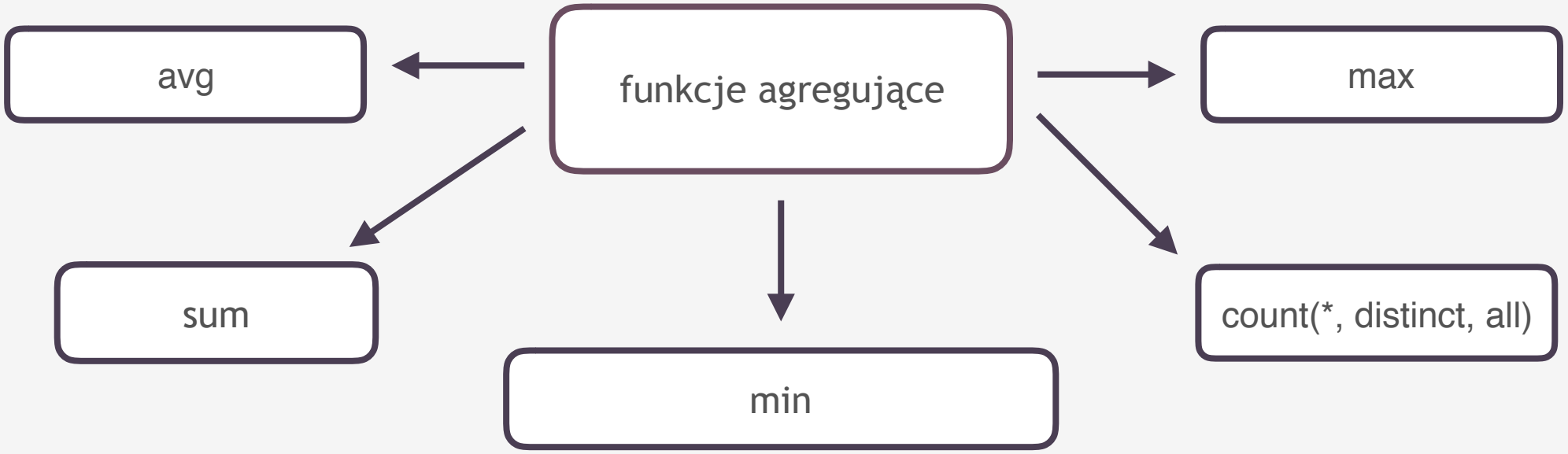
HQL - klauzula group by

Zapytanie zwracające zagregowane wartości może zwracać wyniki pogrupowane w.g. dowolnego atrybutu odpytywanej klasy lub jej składowej:

```
select osoba.obywatelstwo, sum(osoba.wzrost), count(osoba) from Osoba osoba group by
osoba.obywatelstwo
```

```
String hql = "SELECT SUM(E.salary), E.firtName FROM Employee E " +
             "GROUP BY E.firstName";
Query query = session.createQuery(hql);
List results = query.list();
```

HQL - funkcje agregujące



Mapowanie

Rodzaje mapowania

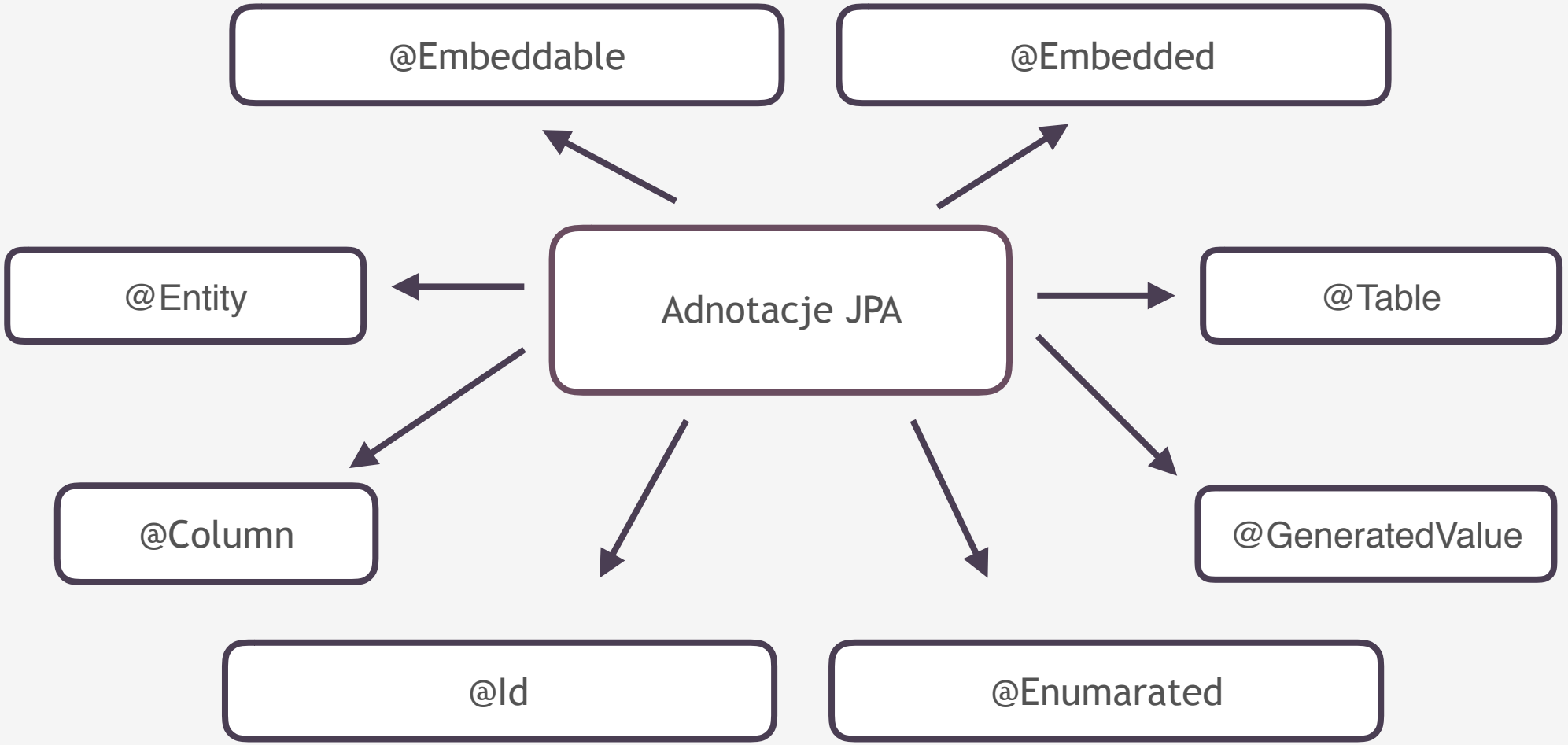
Plik hbm.xml

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
 "-//Hibernate/Hibernate Mapping DTD//EN"
 "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
   <class name="Employee" table="EMPLOYEE">
      <meta attribute="class-description">
         This class contains the employee detail.
      </meta>
      <id name="id" type="int" column="id">
         <generator class="native"/>
      </id>
      <property name="firstName" column="first_name" type="string"/>
      <property name="lastName" column="last_name" type="string"/>
      <property name="salary" column="salary" type="int"/>
   </class>
</hibernate-mapping>
```

Adnotacje

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

    <session-factory>
        <!— database configuration —>
        <mapping class="pl.sdaacademy.model.Author"/>
        <mapping class="pl.sdaacademy.model.Book"/>
    </session-factory>

</hibernate-configuration>
```

Adnotacje


@Embeddable
@Embedded
@Entity
Adnotacje JPA
@Table
@Column
@GeneratedValue
@Id
@Enumarated

Adnotacje JPA - podstawowe użycie

```
@Entity
@Table( name = "books" )
public class Book {

    @Id
    @GeneratedValue(generator="system-uuid")
    @GenericGenerator(name="system-uuid", strategy = "uuid")
    @Column(name = "book_id")
    private String id;

    @Column(name = "book_name")
    private String name;

    @Column(name = "book_type")
    private BookType bookType;

    public Book() {
    }

    //getter and setter

    @Enumerated(EnumType.STRING)
    public BookType getBookType() {
        return bookType;
    }
}
```

```
public enum BookType {

    COMICS("comics"), FANTASY("fantasy"), DEFAULT("default");

    private String type;

    BookType(String type) {
        this.type = type;
    }

    @Override
    public String toString() {
        return type;
    }
}
```

Adnotacje JPA - @Embeddable i @Embedded

```
@Embeddable
public class Animal {
	@Column(name = "name")
	private String name;
	@Column(name = "location")
	private String location;
}
```

```
@Entity
@Table(name = "elephant")
public class Elephant {
	@Id
	@Column(name = "id")
	private int id;
	@Embedded
	@AttributeOverrides({ @AttributeOverride(name = "location", column = @Column(name = "place")) })
	private Animal animal;
```

Adnotacje JPA - @MappedSuperClass

```
@MappedSuperclass
public abstract class BaseEntity {
    @Id
    @GeneratedValue
    private Long id;

    @Version
    private long version;

    // getters/setters etc.
}
```

```
@Entity
@Table(name = "\"user\"")
public class User extends BaseEntity {
    private String username;

    @org.hibernate.annotations.Type(type = "yes_no")
    private boolean isAdmin;

    // constructor/getters/setters etc.
}
```

CRUD

CRUD - select

```
public void listEmployees( ){
   Session session = factory.openSession();
   Transaction tx = null;
   try{
      tx = session.beginTransaction();
      List employees = session.createQuery("FROM Employee").list();
      for (Iterator iterator =
                        employees.iterator(); iterator.hasNext();){
         Employee employee = (Employee) iterator.next();
         System.out.print("First Name: " + employee.getFirstName());
         System.out.print("  Last Name: " + employee.getLastName());
         System.out.println("  Salary: " + employee.getSalary());
      }
      tx.commit();
   }catch (HibernateException e) {
      if (tx!=null) tx.rollback();
      e.printStackTrace();
   }finally {
      session.close();
   }
}
```

CRUD - insert

```
public Integer addEmployee(String fname, String lname, int salary){
    Session session = factory.openSession();
    Transaction tx = null;
    Integer employeeID = null;
    try{
       tx = session.beginTransaction();
       Employee employee = new Employee(fname, lname, salary);
       employeeID = (Integer) session.save(employee);
       tx.commit();
    }catch (HibernateException e) {
       if (tx!=null) tx.rollback();
       e.printStackTrace();
    }finally {
       session.close();
    }
    return employeeID;
 }
```

CRUD - update

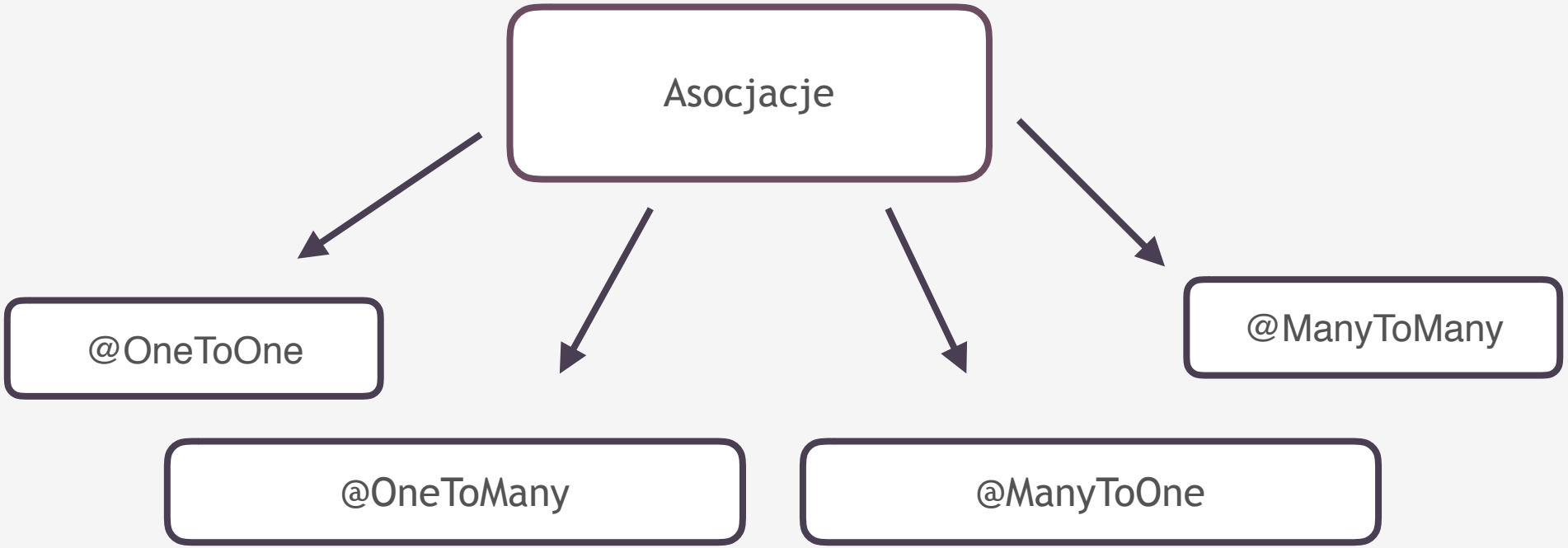
```
public void updateEmployee(Integer EmployeeID, int salary ){
     Session session = factory.openSession();
     Transaction tx = null;
     try{
        tx = session.beginTransaction();
        Employee employee =
                  (Employee)session.get(Employee.class, EmployeeID);
        employee.setSalary( salary );
                        session.update(employee);
        tx.commit();
     }catch (HibernateException e) {
        if (tx!=null) tx.rollback();
        e.printStackTrace();
     }finally {
        session.close();
     }
  }
```

CRUD - delete

```
public void deleteEmployee(Integer EmployeeID){
   Session session = factory.openSession();
   Transaction tx = null;
   try{
      tx = session.beginTransaction();
      Employee employee =
               (Employee)session.get(Employee.class, EmployeeID);
      session.delete(employee);
      tx.commit();
   }catch (HibernateException e) {
      if (tx!=null) tx.rollback();
      e.printStackTrace();
   }finally {
      session.close();
   }
}
```

Relacje

Asocjacje



@OneToOne

```
@Entity
public class Employee{
    @Id  @GeneratedValue(strategy=GenerationType.AUTO)
    private long empId;
…
   @OneToOne
   private Department department;
}

@Entity
public class Department{
    @Id  @GeneratedValue
    private long empId;
…
}
```

@OneToMany lub @ManyToOne

```
@Entity
public class Employee{
    @Id
@GeneratedValue(strategy=GenerationType.AUTO)
    private long empId;
…
   @OneToMany
   private Collection<Project> projects=new
ArrayList<>();
}

@Entity
public class Project{
    @Id
@GeneratedValue(strategy=GenerationType.AUTO)
    private long projectId;
…
}
```

```
@Entity
public class Project{
    @Id  @GeneratedValue(strategy=GenerationType.AUTO)
    private long projectId;
…
   @ManyToOne
   private Employee employee;
}
```

@ManyToMany

```
@Entity
public class Book{
    @Id  @GeneratedValue(strategy=GenerationType.AUTO)
    private long isbn;
…
   @ManyToMany(mappedBy=”books”)
   private Collection<Author> authors=new ArrayList<>();
}

@Entity
public class Author{
    @Id  @GeneratedValue(strategy=GenerationType.AUTO)
    private long authorId;
…
@ManyToMany
   private Collection<Book> books=new ArrayList<>();
}
```

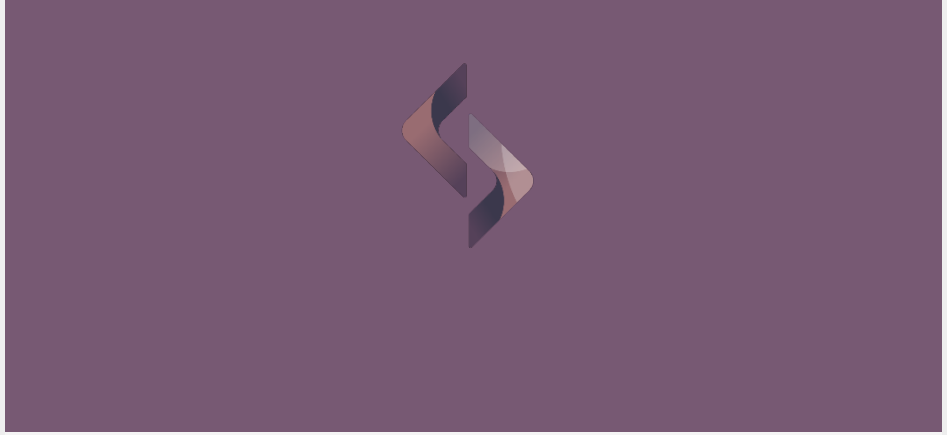
CQRS

CQRS - Command Query Responsibility Segregation

```
public interface IUserQueryService {

    List<User> getUsersList(int page, int size,
String sortDir, String sort);

    String checkPasswordResetToken(long
userId, String token);

    String checkConfirmRegistrationToken(String
token);

    long countAllUsers();

}
```

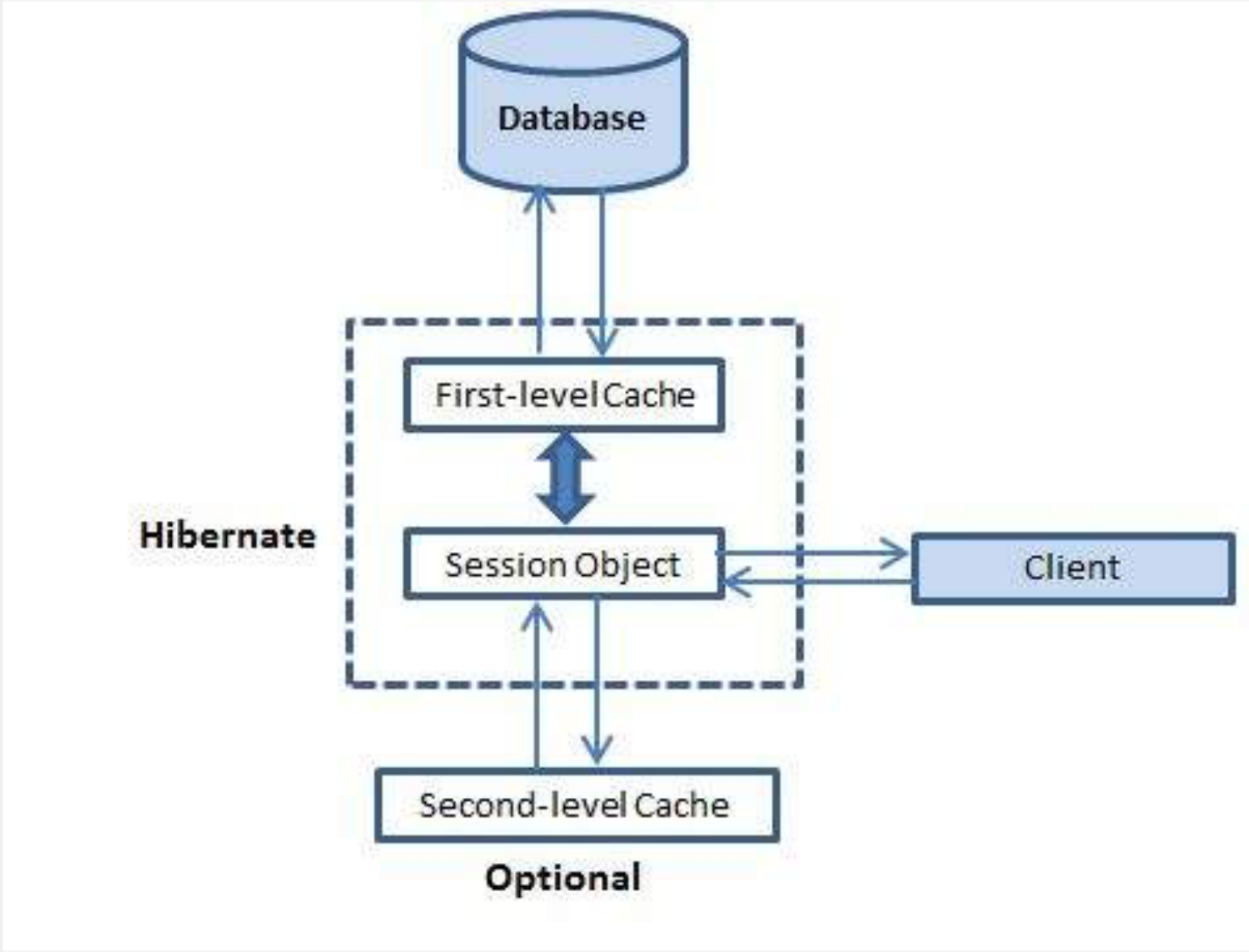
```
public interface IUserCommandService {

    void registerNewUser(String username, String email, String
password, String appUrl);

    void updateUserPassword(User user, String password, String
oldPassword);

    void changeUserPassword(User user, String password);

    void resetPassword(String email, String appUrl);

    void createVerificationTokenForUser(User user, String token);

    void updateUser(User user);

}
```

Cache



Rodzaje cache’u:

- First-level Cache
- Second-level Cache
- Query-level Cache

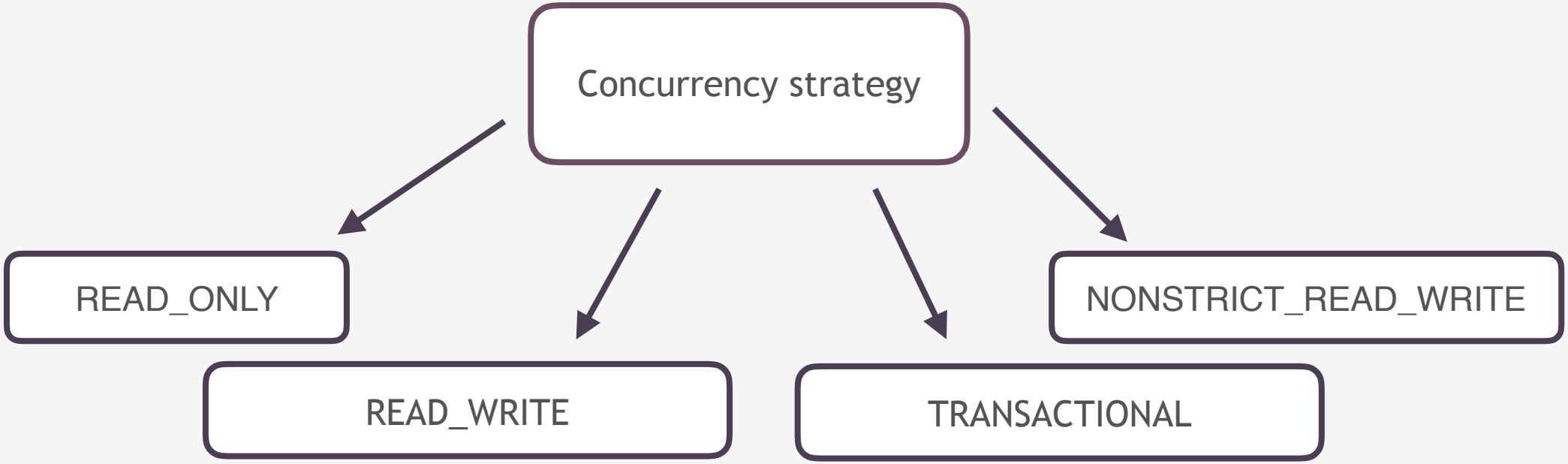


First-level Cache

- Powiązany z obiektem sesji
- Włączony domyślnie
- Brak możliwości wyłączenia go
- Hibernate dostarcza moźliwość wyczyszczenia poszczególnych elementów z pamięci lub całego cache’u
- Brak współdzielenia pamięci pomiędzy sesjami

Second-level Cache

- Wyłączony domyślnie
- Istnieje możliwość konfiguracji po przez pliki konfiguracyjne
- Niezbędne jest określenie strategii współbieżności
- Współdzielony pomiędzy sesjami



Second-level Cache

```
@Entity
@Cacheable
@org.hibernate.annotations.Cache(usage =
CacheConcurrencyStrategy.READ_WRITE)
public class Foo {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(name = "ID")
    private long id;

    @Column(name = "NAME")
    private String name;

    // getters and setters
}
```

```
Foo foo = new Foo();
fooService.create(foo);
fooService.findOne(foo.getId());
int size =
CacheManager.ALL_CACHE_MANAGERS.get(0)
  .getCache("org.baeldung.persistence.model.Foo")
.getSize();
```

Query-level Cache

- Wyłączony domyślnie
- By uruchomić należy ustawić w pliku konfiguracyjnym poniższe pole: hibernate.cache.use_query_cache="true"

```
Session session = SessionFactory.openSession();
Query query = session.createQuery("FROM EMPLOYEE");
query.setCacheable(true);
query.setCacheRegion("employee");
List users = query.list();
SessionFactory.closeSession();
```