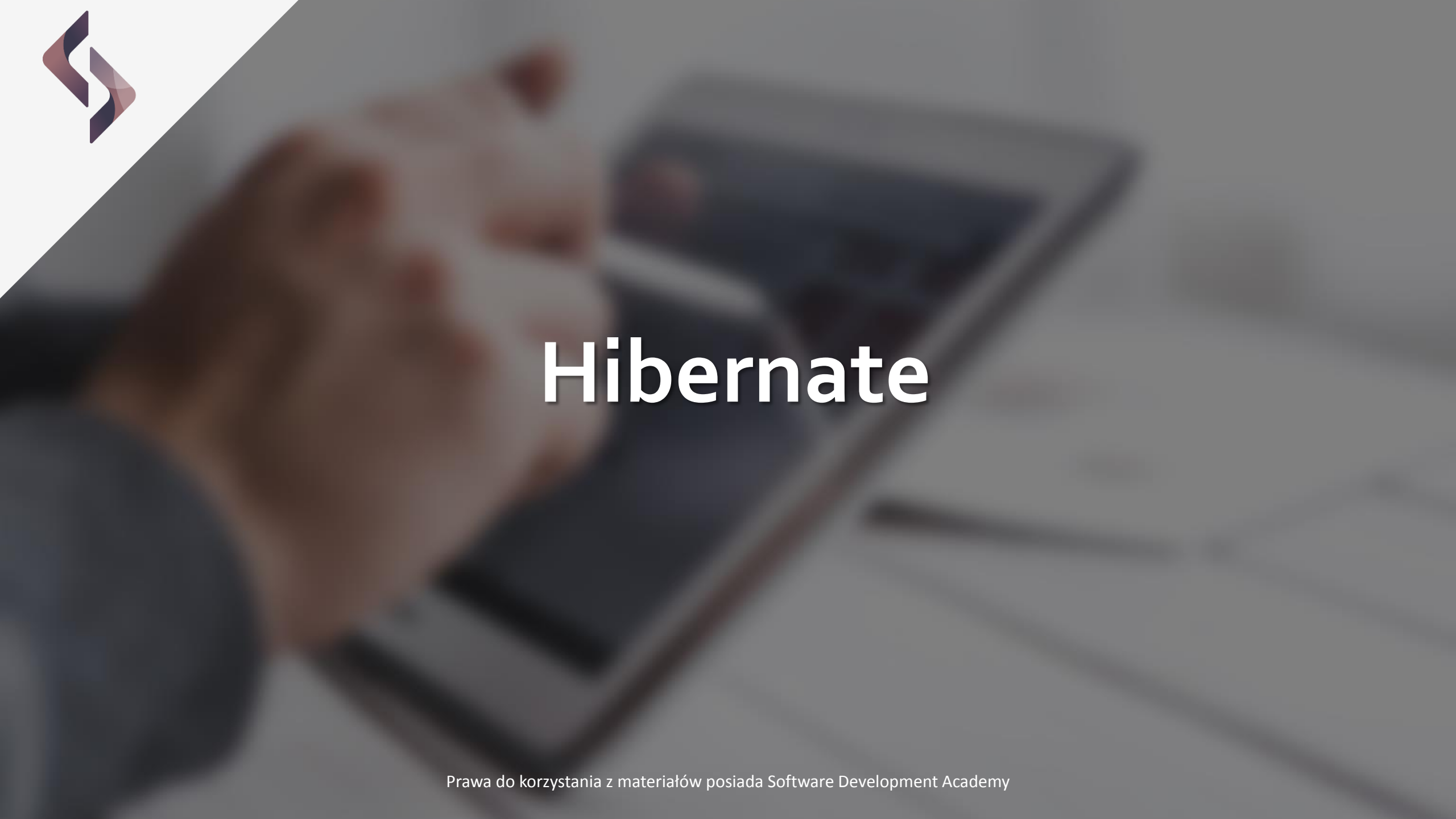


Hibernate

Prawa do korzystania z materiałów posiada Software Development Academy

Prowadzący

Bartosz Andreatto
(**bandreatto@gmail.com**)

programista – Bank Pekao S.A.
https://pl.linkedin.com/in/bartosz-andreatto-02b03413a

HQL

HQL jest bardzo podobny do SQL, **pozwala jednak na wykorzystanie polimorfizmu, asocjacji i innych cech modelu obiektowego w zapytaniach**. Zapytania HQL są **automatycznie tłumaczone na język SQL**, a wyniki zapytania są zwracane w formie referencji do odpowiednich obiektów. **W języku HQL znajdziemy słowa kluczowe, które znajdują się również w języku SQL**, np.:

- klauzule: **SELECT**, **GROUP BY**, **WHERE**, **ORDER BY**, **FROM**
- funkcje agregujące: **AVG**, **SUM**, **MIN**, **MAX**, **COUNT**

Prawa do korzystania z materiałów posiada Software Development Academy

Wykonywanie zapytań HQL

Zapytania HQL możemy wykonywać za pomocą obiektu **EntityManager** i wykorzystując metodę **createQuery**. Metoda ta zwraca obiekt **Query**, który pozwala:

- **wykonać zapytanie za pomocą metody executeUpdate**
- **pobrać zwrócony wynik za pomocą metod getFirstResult** lub **getResultList**
- ustawić parametry zapytania **setParameter**

UWAGA: W przypadku chęci pobrania wyniku zapytania, do metody **createQuery** możemy podać dodatkowy argument będący typem zwracanym.

Prawa do korzystania z materiałów posiada Software Development Academy

FROM

Klauzula FROM używana jest do wskazania klasy, która ma być wykorzystywana w procesie wyszukiwania jej instancji. Jest ona pewnego rodzaju skróconym zapisem zapytania SQL SELECT * FROM tabela. **Pozwala również nadać encji pewien alias** (podobnie jak w języku SQL).

```
FROM nazwaEncji [[AS] alias]
```

UWAGA: **nazwaEncji** w powyższym przykładzie jest **nazwą klasy lub nazwą wskazaną w adnotacji @Entity**. Jeżeli nazwa tabeli jest wskazana w adnotacji @Table, w zapytaniach HQL **nadal wykorzystujemy nazwę klasy**.

UWAGA: Język HQL jest ***case insensitive**.

Prawa do korzystania z materiałów posiada Software Development Academy

HQL - przykład

```java
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;

@Entity(name = "cars")
@Data
@NoArgsConstructor
@AllArgsConstructor
public class Car {

  @Id
  @GeneratedValue
  private Long id;

  private String modelName;

  private String producer;

  private Double engineVolume;

  public Car(String modelName, String producer, Double engineVolume) {
    this.modelName = modelName;
    this.producer = producer;
    this.engineVolume = engineVolume;
  }
}
```

Prawa do korzystania z materiałów posiada Software Development Academy

FROM

Chcąc pobrać wszystkie obiekty typu Car, wykonujemy zapytanie:

```
List<Car> cars = entityManager.createQuery("FROM cars", Car.class).getResultList();
```

Kolejny przykład pokazuje to samo zapytanie **z wykorzystaniem aliasu**.

```
List<Car> cars = entityManager.createQuery("FROM cars c, Car.class).getResultList();
```

Prawa do korzystania z materiałów posiada Software Development Academy

SELECT

Klauzula **SELECT** działa podobnie jak FROM, z tą różnicą, że pozwala **na określenie, które z atrybutów obiektu powinny zostać zwrócone** przez zapytanie, tzn.:

```
SELECT nazwaPolaKlasyA, nazwaPolaKlasyB FROM nazwaEncji [[AS] alias]
```

Poniższy przykład pobiera **listę co najwyżej 3 producentów**:

```
List<String> producers = entityManager.createQuery("SELECT c.producer FROM cars c", String.class)
        .setMaxResults(3)
        .getResultList();
```

Prawa do korzystania z materiałów posiada Software Development Academy

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, jednakże, **zamiast wykorzystywać nazwy kolumn, wykorzystujemy nazwy pól klas**:

```
FROM nazwaEncji [[AS] alias] WHERE nazwaEncji[alias]=wskazana_wartość
```

Prawa do korzystania z materiałów posiada Software Development Academy

WHERE

Chcąc pobrać wszystkie obiekty typu **Car**, którego model **modelName** ma wartość **Aygo**, wykonujemy zapytanie:

```
List<Car> cars = entityManager.createQuery("FROM cars c WHERE c.modelName = 'Aygo'", Car.class).getResultList();
```

Poniższy przykład pobiera listę producentów, którzy oferują samochód o nazwie Aygo:

```
List<String> producers = entityManager.createQuery("SELECT c.producer FROM cars c WHERE c.modelName = 'Aygo'", String.class)
        .getResultList();
```

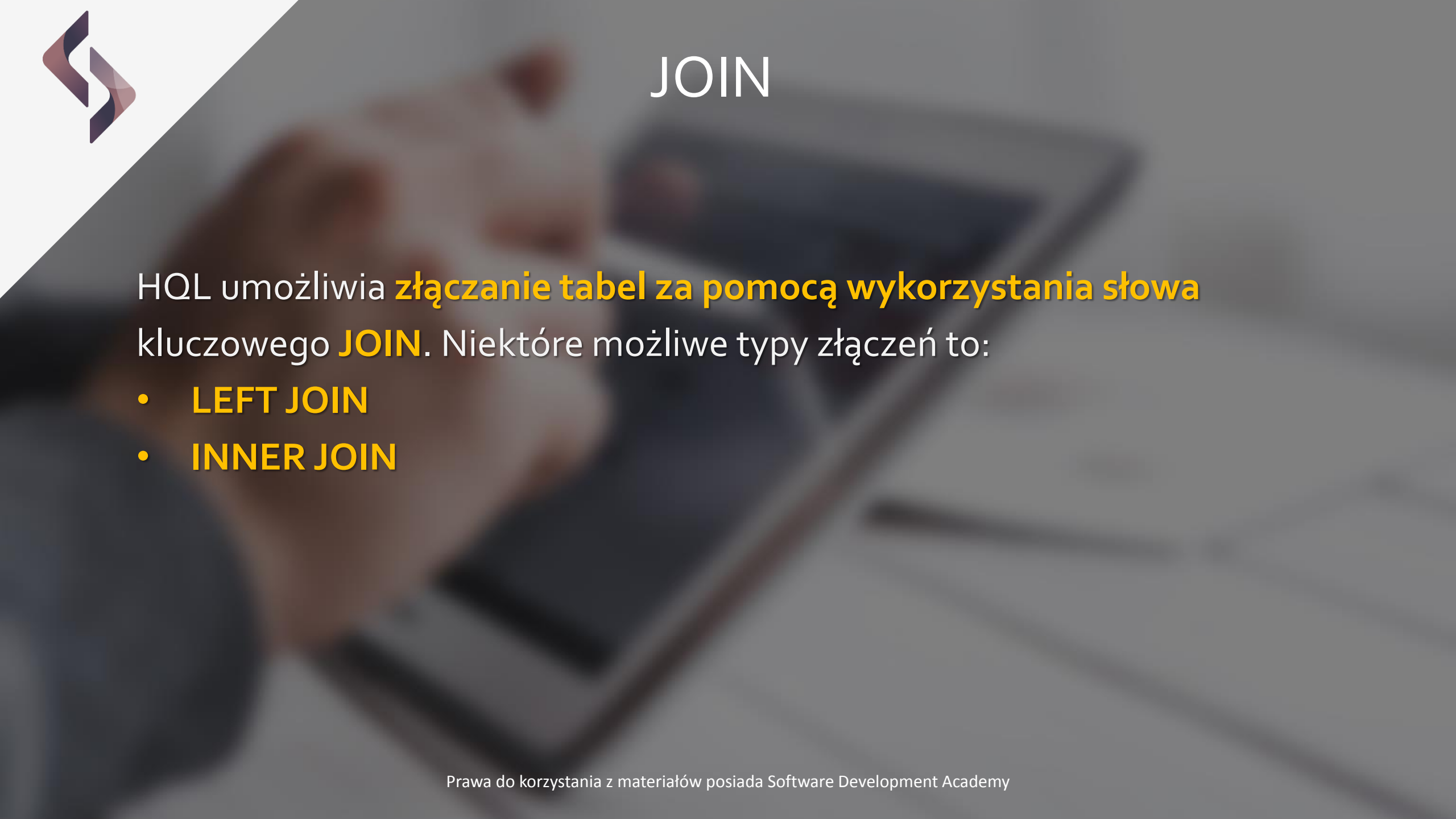
Prawa do korzystania z materiałów posiada Software Development Academy

ORDER BY

ORDER BY umożliwia **porządkowanie zwróconych rezultatów** w ramach dowolnego atrybutu zwracanego obiektu. Tak jak w przypadku natywnego SQLa, **ORDER BY umieszczamy na końcu zapytania**. Wyniki możemy posortować rosnąco za pomocą słowa kluczowego **ASC** lub malejąco wykorzystując **DESC**.

```
List<Car> cars = entityManager.createQuery("SELECT c FROM cars c ORDER BY c.engineVolume DESC", Car.class).getResultList();
```

Prawa do korzystania z materiałów posiada Software Development Academy



JOIN

HQL umożliwia **złączanie tabel za pomocą wykorzystania słowa** kluczowego **JOIN**. Niektóre możliwe typy złączeń to:

- **LEFT JOIN**
- **INNER JOIN**

Prawa do korzystania z materiałów posiada Software Development Academy

JOIN

```
Car car = entityManager.createQuery("SELECT c FROM cars c left join c.wheels w WHERE w.diameter > 15", Car.class).getSingleResult();
```

Ponadto, klauzula **with** umożliwia **określanie warunku dla łączonej tabeli**, np.:

```
List<Car> cars = entityManager.createQuery("SELECT c FROM cars c left join c.wheels as w with w.producer = 'Pirelli' WHERE w.diameter > 15", Car.class).getResultList()
```

UWAGA: HQL pozwala również wykorzystywać słowo kluczowe fetch, które pobierze obiekty, które mogą być pobierane w sposób 'lazy'.

Prawa do korzystania z materiałów posiada Software Development Academy

Prepared statement

HQL, podobnie jak **JDBC**, pozwala na wykonywanie zapytań parametryzowanych. Wartość każdego z parametrów powinna mieć dowolną, unikalną nazwę **poprzedzoną znakiem :.**

```
List<Car> cars = entityManager.createQuery("SELECT c FROM cars c WHERE c.modelName = :modelName AND c.engineVolume = :engineVolume", Car.class)
        .setParameter("modelName", "Corolla")
        .setParameter("engineVolume", 2.0)
        .getResultList();
```

Prawa do korzystania z materiałów posiada Software Development Academy

Funkcje agregujące

HQL, podobnie jak SQL, pozwala wykorzystać funkcje agregujące, np.:

```
SELECT AVG(p.price), SUM(p.quantity), MAX(p.quantity), COUNT(p) FROM Product p
```

Prawa do korzystania z materiałów posiada Software Development Academy

Dziękuję

Prawa do korzystania z materiałów posiada Software Development Academy