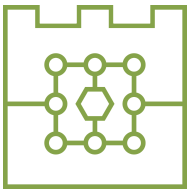

Programowanie w języku Java

Laboratorium nr 4

Politechnika Krakowska
Wydział Informatyki i Telekomunikacji
Katedra Informatyki

2024

Omówienie narzędzia Hibernate

Oficjalna dokumentacja:
https://hibernate.org/orm/what-is-an-orm/
Hibernate jest najpopularniejszą biblioteką służącą do mapowania obiektowo-relacyjnego w Javie (ORM / Object Relational Mapping).
Hibernate pozwala automatycznie mapować obiekty Javy na wiersze w bazie danych oraz odczytywać rekordy z bazy danych i automatycznie tworzyć z nich obiekty. Na dobrą sprawę wykorzystując Hibernate teoretycznie nie musimy mieć większego pojęcia o poprawnym konstruowaniu zapytań w języku SQL, ponieważ będą one budowane za nas.

Konfiguracja Hibernate

Można to zrobić na 3 sposoby:

- Utworzyć plik właściwości
- Utworzyć plik hibernate.cfg.xml
- Użyć metod komponentu bean konfiguracji.

W tym laboratorium omówimy tworzenie pliku *hibernate.cfg.xml*
Ustawienia pliku konfiguracyjnego można podzielić na 3 części:

1. Określ typ DBMS
   - Określ dialekt (typ DBMS), na przykład Oracle 9.0.
   - Podaj nazwę sterownika JDBC do pracy z tym DBMS.
2. Określ dane do autoryzacji w bazie danych
   - Adres bazy danych.
   - Nazwa użytkownika.
   - Hasło.
3. Konfigurowanie silnika Hibernate
   - `hibernate.show_sql` - Ustawienie, dzięki któremu Hibernate zduplikuje w konsoli wszystkie żądania, które wykonuje.
   - `hbm2ddl.auto` - Ustawienie, dzięki któremu Hibernate w razie potrzeby zmieni strukturę bazy danych.

Pobranie `SessionFactory` za pomocą pliku hibernate.cfg.xml
Przykład kodu:

```
1     SessionFactory sessionFactory = new
          ↪ Configuration().configure().buildSessionFactory();
```

Kiedy konfigurujemy obiekt SessionFactory w Hibernate, ten SessionFactory sprawdza, czy wszystkie wymagane tabele z wymaganymi typami kolumn istnieją w bazie danych. Dlatego należy określić, gdzie szukać klas jednostek:
Przykład kodu:

```
1     <mapping class="class-name" />
```

Mapowanie Encji w Hibernate

Każda klasa oznaczona @Entity powinna mieć klucz główny oznaczony @Id. Hibernate wymaga, aby encje miały bezparametrowy konstruktor (może być domyślny). Wszystkie pola w encji są domyślnie mapowane na kolumny, chyba że użyjesz @Transient, aby wykluczyć pole z mapowania.

Adnotacja @Entity, @Table

Jeśli chcesz zmapować swoją klasę do tabeli w bazie danych, to w tym celu musisz dodać do niej aż trzy adnotacje.

Najpierw musisz dodać adnotację do klasy `@Entity`. Ta adnotacja przed klasą powie Hibernate, że nie jest to tylko klasa, ale specjalna klasa, której obiekty muszą być przechowywane w bazie danych.

Przykład:

```
@Entity
class User {
    public int id;
    public String name;
    public int level;
    public Date createdDate;
}
```

To wystarczy, aby Hibernate traktował obiekty tej klasy w szczególny sposób.

Druga adnotacja to `@Table`. Za jej pomocą można ustawić nazwę tabeli w bazie danych, z którą ta klasa będzie powiązana.

Przykład:

```
@Entity
@Table(name="user")
class User {
    public int id;
    public String name;
    public int level;
    public Date createdDate;
}
```

Jeśli nazwa klasy i nazwa tabeli są takie same, adnotację `@Table` można pominąć.

Jeśli Twoja aplikacja działa z tabelami z kilku schematów jednocześnie, musisz określić, w którym konkretnym schemacie znajduje się tabela:

```
@Entity
@Table(name="user", schema="test")
class User {
    public int id;
    public String name;
    public int level;
```

```
    public Date createdDate;
}
```

Tak, wszystko jest tak proste, jak się wydaje.

Adnotacja @Column

Drugą ważną rzeczą, którą należy wiedzieć, jest mapowanie kolumn tabeli na pola klas. W najprostszej wersji Hibernate po prostu mapuje pola Twojej klasy encji na kolumny żądanej tabeli.

Jeśli chcesz kontrolować wszystkie niuanse mapowania, możesz użyć adnotacji `@Column`.

Przykład:

```
@Entity
@Table(name="user")
class User {
    @Column(name="id")
    public Integer id;

    @Column(name="name")
    public String name;

    @Column(name="level")
    public Integer level;

    @Column(name="created_date")
    public Date createdDate;
}
```

Adnotacja `@Column` ma różne parametry, poniżej rozważymy najpopularniejsze z nich:

#	Nazwa atrybutu	Opis
1	`name`	Ustawia nazwę kolumny tabeli dla pola klasy.
2	`unique`	Wszystkie wartości pól muszą być unikalne.
3	`nullable`	Określa, czy pole może być puste (`null`).
4	`length`	Maksymalna długość pola (dotyczy stringów).

Dodajmy pewne ograniczenia do pól naszej klasy `User Entity`:

- nazwa użytkownika musi być unikalna i nie dłuższa niż 100 znaków,
- poziom może być zerowy,
- data utworzenia nie może mieć wartości `null`.

Przykład:

```
@Entity
@Table(name="user")
class User {
    @Column(name="id")
    public Integer id;

    @Column(name="name", unique=true, length=100)
    public String name;

    @Column(name="level", nullable=true)
    public Integer level;

    @Column(name="created_date", nullable=false)
    public Date createdDate;
}
```

Przy próbie zapisania obiektu do bazy danych Hibernate sprawdzi ograniczenia i, jeśli zostaną naruszone, zgłosi wyjątek.

Adnotacja @Id

`@Id` to adnotacja służąca do oznaczenia klucza podstawowego w tabeli. Wystarczy podać tę adnotację dla żądanego pola, a Hibernate zajmie się resztą.

Przykład:

```
@Entity
@Table(name="user")
class User {
    @Id
    @Column(name="id")
    public Integer id;

    @Column(name="name")
    public String name;

    @Column(name="level")
    public Integer level;

    @Column(name="created_date")
    public Date createdDate;
}
```

Jeśli chcesz, aby Hibernate automatycznie generował identyfikatory dla Twoich obiektów, możesz dodać adnotację `@GeneratedValue`.

Przykład:

```java
@Entity
@Table(name="user")
class User {
    @Id
    @GeneratedValue
    public Integer id;

    @Column(name="name")
    public String name;

    @Column(name="level")
    public Integer level;

    @Column(name="created_date")
    public Date createdDate;
}
```

Zadania

Konfiguracja Hibernate

Skonfigurować Hibernate do pracy z bazą danych PostgreSQL.

1. Pobierz i zainstaluj PostgreSQL
   Link do instalacji: https://www.postgresql.org/download/

2. Dodaj zależności **Hibernate i PostgreSQL** do pliku *pom.xml*
   * Aby zależności zostały pobrane, należy przeładować MAVENA.
   * Najlepiej szukać zależności w oficjalnym repozytorium Maven:
   https://mvnrepository.com/artifact/org.postgresql/postgresql/42.7.2

3. Aby nawiązać połączenie z bazą danych, należy w katalogu *src/main/resources* utworzyć oraz odpowiednio skonfigurować plik *hibernate.cfg.xml*.
   Przykładowa konfiguracja dla MySQL: https://www.tutorialspoint.com/hibernate/hibernate_configuration.htm

4. Następnie dodaj odpowiednie adnotacje Hibernate do klas, aby umożliwić ich mapowanie na tabele w bazie danych.
   (Należy zaimportować pakiet *jakarta.persistence.**, który zawiera potrzebne adnotacje do definiowania mapowania klas na tabele w bazie danych.)

5. Stwórz klasę DAO (Data Access Object) *ShapeDAO*, aby zarządzać zapisami i odczytami z bazy danych.

6. Sprawdź, czy baza danych oraz jej tabele zostały poprawnie utworzone, a zapis i odczyt danych z bazy przebiegają bez błędów.

7. Dodaj testy jednostkowe, aby zweryfikować poprawność interakcji z bazą danych:

(a) Skonfiguruj bazę danych w pamięci (np. H2) lub rzeczywistą bazę PostgreSQL do testów.

(b) Dodaj testy, np.:

- czy tabela *Shape* została poprawnie utworzona;
- czy operacje zapisu, odczytu, aktualizacji i usuwania działają bez błędów;