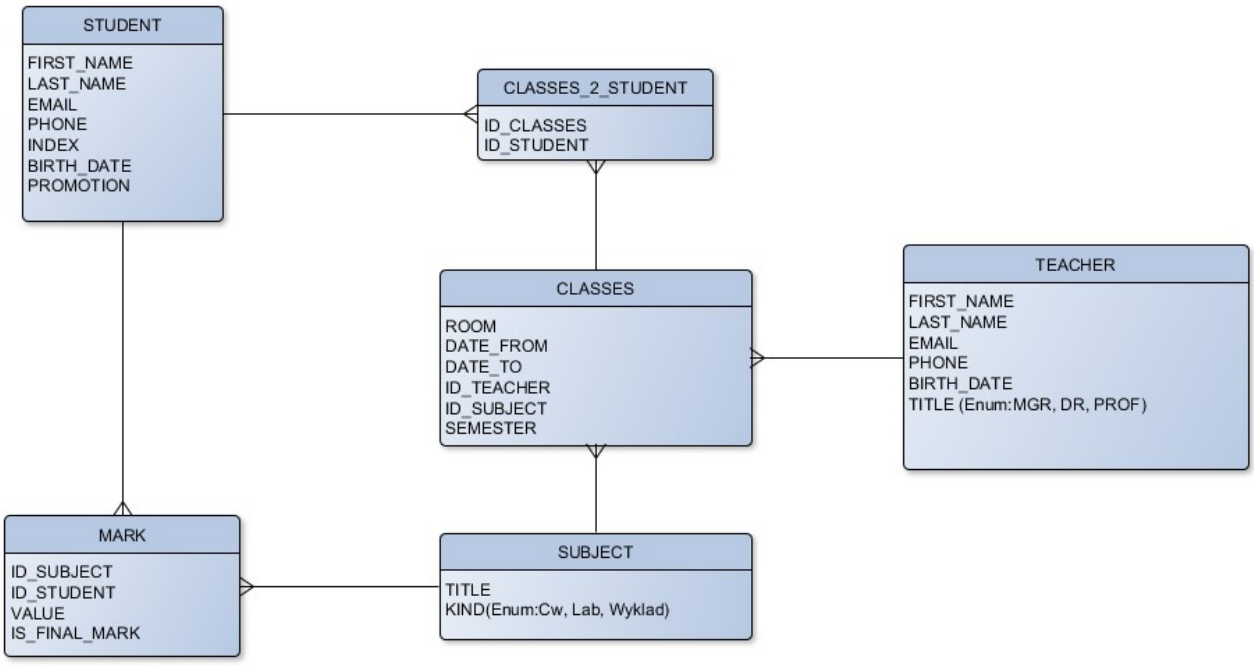
Narzędzie mapowania obiektowo-relacyjnego Hibernate

Ćwiczenia

Cykl ćwiczeń tworzy tematycznie wspólną całość. Ćwiczenia mają na celu nauczenie szkolonego praktycznego zastosowania biblioteki Hibernate. W tym celu kolejne kroki ćwiczeń opisują budowę od zera. Szkolony tworzyć będzie przede wszystkim fragmenty aplikacji odpowiedzialne za interakcję z bazą danych.
Budowana aplikacja przeznaczona jest dla pracowników dziekanatu. Pomaga wyszukiwać i modyfikować dane dotyczące studentów, nauczycieli i zajęć.



Ćwiczenie lab_01

Cel: Konfiguracja hibernate i przećwiczenie zachowań parametru dll2auto

Zadanie:

Na komputerze została zainstalowana baza danych PostgreSQL oraz PgAdmin. Należy uruchomić PgAdmin i podłączyć się do bazy danych „postgres" - użytkownik i hasło taki sam jak nazwa bazy. Baza danych nie powinna zawierać żadnych tabel.

W projekcie deneary-blc w pakiecie model definiujemy klasę Teacher i adnotujemy ją wymagalnymi adnotacjami.(Pomijamy typ wyliczeniowy enum oraz date)

Następnie uzupełniamy HibernateUtil który zawiera kod tworzący fabrykę sesji hibernate oraz plik hibernate.cfg-postgres.xml (pamiętaj o dodaniu klasy Teacher w konfiguracji). Na końcu uruchamiamy test `Hbm2ddlTest` który powinien nam utworzyć sesję. W dokumentacji hibernate (http://docs.jboss.org/hibernate/orm/4.2/manual/en-US/html_single/) odnajdujemy jakie wartości może przyjąć parametr `hbm2ddl.auto` oraz eksperymentujemy ze wszystkimi wariantami. Należy zwrócić uwagę jak i kiedy są tworzone/aktualizowane tabele. Jakie idą SQL, czy dane z tabel są czyszczone

Dodatkowo 1: Oznaczyć kolumnę email jako unikalną i wymaganą
Dodatkowo 2: Spróbować skonfigurować pola firstName oraz lastName tak aby pochodziły z drugiej tabeli np. PersonData

Ćwiczenie lab_02

Cel: Praca z obiektem Teacher – CRUD

Zadanie:
Zaimplementować testy w TeacherTest oraz odpowiednią funkcjonalność w klasie TeacherDao należy zwrócić uwagę w jaki sposób została zaimplementowana klasa `BaseTest` oraz w jaki sposób sesja trafia do obiektu `TeacherDao`.

Uwaga: Puszczając każdy testy zwróć uwagę na ilość SQL które idą na bazę danych

Dodatkowo 1: Sprawdzić jak zachowa się hibernate jak nie podamy mu wymaganego adresu email, albo adresu już istniejący

Ćwiczenie lab_03

Cel: Enumy

Zadanie:
Do klasy teacher domapować kolumnę TITLE, enuma którego należy również stworzyć porónać dwa sposoby działania Ordinal oraz String.

Ćwiczenie lab_04

Cel: Daty

Zadanie:
Do klasy teacher domapować kolumne BIRTH_DATE, sprawdzić różne ustawienia TemporalType - co jest zapisywane w bazie danych i za pomocą jakiego typu. Jakie jest domyślne zachowanie Hibernate bez adnotacji temploral.

Ćwiczenie lab_05

Cel: Porównanie implementacji nadawania klucza głównego

Zadanie:
Porównać strategie nadawania klucza głownego. W tescie IdTest dodać 10.000 obiektów Teacher. Który strategia jest najwydajniejsza biorąc pod uwagę czas wykonania testu.

Ćwiczenie lab_06

Cel: Przećwiczenie języka zapytań HQL – poziom podstawowy

Zadanie:
Uzupełnić test `HQLTest`, implementując wymagane zapytania. Aby praca była łatwiejsza należy napełnić bazę danych danymi testowymi. Można to zrealizować za pomocą klasy `DataImporter` + dodatkowo należy przestawić *`hbm2ddl.auto`* na **update** aby nie ginęły dane pomiędzy testami

Ćwiczenie lab_07

Cel: Przećwiczenie zapytań Criteria API – poziom podstawowy

Zadanie:
Uzupełnić test `CriteriaTest`, implementując wymagane zapytania. Aby praca była łatwiejsza dodaj do bazy danych na stałe np. 10 obiektów Teacher. Można to zrealizować za pomocą klasy `DataImporter`

Ćwiczenie lab_08

Cel: Przećwiczenie zapytań BY Example – poziom podstawowy

Zadanie:
Uzupełnić test `QueryByExampleTest`, implementując wymagane zapytania. Aby praca była łatwiejsza dodaj do bazy danych na stałe np. 10 obiektów Teacher. Można to zrealizować za pomocą klasy `DataImporter`

Ćwiczenie lab_09

Cel: Zdefiniowanie relacji między obiektami

Zadanie:
Zdefiniować wszystkie Klasy Encji, DAO oraz zdefiniować wszystkie relację pomiędzy obiektami. Zaimplementować testy RelationsTest.

Dodatkowo1: Zastanowić się jaka logika powinna się znaleźć w DAO np. `ClassesDao.addStudentToClasses.`

Dodatkowo2: Czy obiekty DAO nie wyglądają podobnie...co można z tym zrobić?

Ćwiczenie lab_10

Cel: Strategia pobierania relacji LAZY/EAGER

Zadanie:
Na adnotacjach dotyczących relacji OneToMany,ManyToOne itp. istnieje atrybut fetch (LAZY, EAGER), który decyduje w jakiś sposób mają zostać zaczytane relacje. Zwrócić uwagę na SQL'e. Podczas pisania testów oprzyj się o dane istniejące już w bazie np. o studenta o indeksie **"Index 00"**.

W klasie `RelationsLazyTest` zdefiniować metody które:

- `should_read_student_and_sum_all_marks` – pobiera studenta wraz z jego ocenami – jedno zapytanie
- `should_read_student_and_subject_of_mark -` pobiera studenta wraz z jego ocenami oraz przedmiotami, których dotyczą te oceny – jedno zapytanie, przed włączeniem EAGER na relacji Subject sprawdzić jak działa adnotacja @BatchSize na tej właśnie klasie

- `should_read_student_marks_classes -` pobiera studenta wraz z jego ocenami oraz zajęciami – jedno zapytanie

- `should_read_classes – zaimplementujmy jako classesDao.get(1L). Dlaczego podczas pobierania obiektu classes automatyczne idzie SQL zawierający left outer join na tabele subject?? A czemu outer a nie np. inner. Pobawić się parametrem @ManyToOne(optional, fetch)`

Ćwiczenie lab_11

Cel: Implementacja Equals oraz HashCode

Zadanie:
Wprowadzić klasę BaseEntity po której dziedziczą wszystkie encje i zaimplementować w niej equals oraz hashCode oparty o wartość klucza głównego – Long. Skorzystać z testów EqualsTest

Ćwiczenie lab_12

Cel: Implementacja obiektów osadzonych

Zadanie:

Wprowadzić klasę SalaryInfo która będzie zawierać salary, lastSalaryRise, currentCredit. Wykorzystać stworzoną klasę w Teacher. Należy pamiętać o metodach equals oraz hashCode. Zaimplementować `EmbeddedTest`

Ćwiczenie lab_13

Cel: Deklaracja kaskad na relacjach

Zadanie:

Zdefiniować kaskadę dzięki której oceny dodawane do kolekcji ocen w studencie zostaną zapisane w bazie.

Zdefiniować kaskadę dzięki której usuwając studenta automatycznie usuniemy wszystkie jego oceny.

Ćwiczenie lab_14

Cel: Dziedziczenie w encjach

Zadanie:

Oznaczyć klasę BaseEntity jako @MappedSuperClass oraz dodać do niej atrybut Date lastModified, który oznacza datę ostatniej modyfikacji rekordu.

Ćwiczenie lab_15

Cel: Przećwiczenie języka zapytań HQL – named query

Zadanie:

Za pomocą mechanizmu NamedQuery stworzyć następujące zapytania:

1. Znaleźć wszystkich studentów którzy mają co najmniej 5 ocen – funkcja size vs podzapytanie
2. Znaleźć wszystkich studentów którzy mają co najmniej 2 oceny – pobrać 10 rekordów pomijając pierwsze 10
3. Wyliczyć średnią ocen studenta
4. Wyliczyć średnie ocen studenta w rozbiciu na przedmioty (wynik lista tablic, gdzie w tablicy 0=Subject,1=Double) posortować od najlepszej do najgorszej

5. Znaleźć najbardziej zapracowanego nauczyciela (najwięcej zajęć) w danym okresie dataOd –dataDo
6. Znaleźć nauczycieli którzy posiadają tytuł Dr lub Prof.(jako parametr wejściowy) i nie prowadzili jeszcze żadnych zajęć
7. Mając podanego studenta oraz nauczyciela znaleźć zajęcia na których się spotkają (przedmiot, dataOd,dataDo, miejsce/room) – zastosować specjalną klasę dla wyniku zapytania np. MeetingInfo strona 142. Czyli wynikiem query ma być List<MeetingInfo>
8. Promocją 1: aktualizacja wszystkich ocen studentów o +1;
9. Promocja 2: Usuwamy wszystkie oceny mniejsze niż 2
10. Promocja 3: Dodajemy każdemu studentowi 1 ocenę 5 z dowolnego przedmiotu

Dodatkowo 1: Przyjmujemy, że kolekcja markSet w student jest LAZY bo tak chce większość programistów. Napisać HQL który wyciągnie obiekt Student już z załadowaną kolekcją -FETCH JOIN

Ćwiczenie lab_16

Cel: Criteria API

Zadanie: Wyciągnąć pierwszych dziesięciu studentów(sort alfabetyczny) którzy dostali ocenę co najmniej 5 ze wskazanego przedmiotu

Ćwiczenie lab_17

Cel: SQL API

Zadanie: Napisać algorytm który wyszuka studentów oraz nauczycieli którzy mają tak samo na nazwisko.

Dodatkowo 1: Co jeśli przed wykonaniem tego zapytania pobierzemy dowolnego studenta i zmienimy mu nazwiska tak aby miało wpływ na wynik zapytania SQL

Ćwiczenie lab_18

Cel: :-/

Zadanie: W studencie znajduje się indeks unikalny na pole „index”. Wykonać zamianę indeksów. Czyli przenieść numer indeksu X do Y, a z Y do X

Ćwiczenie lab_19

Cel: Blokowanie pesymistyczne

Zadanie:

1. Zaimplementować metodę która pobierze studenta od edycji (pobierze dane + założy blokadę pesymistyczną (strona 57)

2. Zaimplementować metodę która zablokuje do edycji wyniki zapytania dla pierwszych dziesięciu studentów (sort po indexie)

Ćwiczenie lab_20

Cel: Blokowanie optymistyczne

Zadanie: W klasie BaseEntity pole lastModified oznaczyć adnotacją @Version przeanalizować działanie(sqle) w sytuacji aktualizacji rekordu Student

UWAGA: Aby ten algorytm działał poprawnie wymagane jest ponowne uruchomienie DataImportera aby pole lastModified uzyskało wartość

Ćwiczenie lab_21

Cel: Optymalizacja aplikacji

Zadanie: W dostarczone aplikacji WWW znajdują się 3 funkcjonalności z poniższymi problemami. Problemy należy zidentyfikować i rozwiązać:

1. Menu: Marks - funkcja wyszukiwania ocen studenta po indeksie
    1. Nie działa wyszukiwanie
    2. Kiedyś działało ale wolno
2. Menu: Count Avg marks – funkcja podlicza średnią ocen dla każdego studenta
    1. Strasznie wolno działa (check)
    2. Zwraca złą liczbę wyników powinno być tyle ile studentów (tree + criteria)
3. Menu: Add final marks – funkcja wystawia oceny końcowe
    1. Czas oczekiwanie na wykonanie funkcji większy niż 5min