

Bazy Danych 1 – Projekt

Prowadzący:

Dr inż. Dariusz Jankowski

Autorzy:

Emilia Augustyn, 241248

Mateusz Śliwka, 241375

Joanna Komorniczak, 241245

!UWAGA!

Szanowny Panie Doktorze, pełny projekt tj. aplikacja oraz zrzut bazy danych i plik .jar znajduje się w naszym grupowym repozytorium na GitHub pod adresem

<https://github.com/mateusz-sliwka/SzkolaJazdy-BD1>

Na ePortalu przesyłamy dokumentację. Wszystkie pliki takie jak kod źródłowy aplikacji, export bazy z tabelami obsługującymi sekwencje Hibernate oraz dokumentacja w pliku edytowalnym znajdują się na GitHubie.

1. Wstęp teoretyczny

1.1 Podstawy relacyjnych baz danych

Relacyjna baza danych

Baza danych jest zorganizowanym zbiorem danych (informacji), który przechowywany jest zazwyczaj w formie elektronicznej. Organizacja danych polega na ich odpowiednim podzieleniu i pogrupowaniu według poszczególnych pól, rekordów a także i plików, co ułatwia późniejsze pozyskiwanie, przetwarzanie i wprowadzanie informacji.

Istnieje wiele rodzajów baz danych. Obecnie najbardziej popularnym rodzajem jest **relacyjna baza danych**. Struktura tej bazy oparta jest na obiektach zwanych tabelami (relacjami), pomiędzy którymi definiuje się różne związki. Dane zapisywane są w postaci krotek. Krotki mają swoje atrybuty, a każda krotka zapisana jest w relacji.

Postulaty Codd'a

Podstawą tego modelu stała się praca opublikowana w 1970 r. przez E.F Codd'a - *A Relational Model of Data for Large Shared Data Banks*. Zauważył on, że zastosowanie struktur i procesów matematycznych w zarządzaniu danymi mogłoby rozwiązać wiele problemów trapiących współczesne modele. Relacyjny model bazy danych oparty jest o algebrę relacji.

Postulaty Codd'a:

- **postulat informacyjny** - dane są reprezentowane jedynie przez wartości atrybutów w wierszach tabel (w krotkach)
- **postulat dostępu** - każda wartość w bazie danych jest dostępna poprzez podanie nazwy tabeli, atrybutu i wartości klucza podstawowego (głównego)
- **postulat dotyczący wartości NULL** - dostępna jest specjalna wartość `NULL` dla reprezentacji zarówno wartości nieokreślonej, jak i nieadekwatnej, inna od wszystkich i podlegająca przetwarzaniu
- **postulat dotyczący katalogu** - wymaga się, aby system obsługiwał wbudowany katalog relacyjny z bieżącym dostępem dla uprawnionych użytkowników używających języka zapytań
- **postulat języka danych** - system musi dostarczać pełny język przetwarzania danych, który może być używany zarówno w trybie interaktywnym, jak i w obrębie programów, obsługuje operacje definiowania danych, operacje manipulowania danymi, ograniczenia związane z bezpieczeństwem i integralnością oraz operacje zarządzania transakcji
- **postulat modyfikowalności perspektyw** - system musi umożliwiać modyfikowanie perspektyw, o ile jest ono semantycznie realizowalne.

- **postulat modyfikowalności danych** - system musi umożliwiać operacje modyfikacji danych, musi obsługiwać operacje `INSERT`, `UPDATE` oraz `DELETE`
- **postulat fizycznej niezależności danych** - zmiany fizycznej reprezentacji danych i organizacji dostępu nie wpływają na aplikacje
- **postulat logicznej niezależności danych** - zmiany wartości w tabelach nie wpływają na aplikacje
- **postulat niezależności więzów spójności** - więzy spójności są definiowane w bazie i nie zależą od aplikacji
- **postulat niezależności dystrybucyjnej** - działanie aplikacji nie zależy od modyfikacji i dystrybucji bazy
- **postulat bezpieczeństwa względem operacji niskiego poziomu** - operacje niskiego poziomu nie mogą naruszać modelu relacyjnego i więzów spójności

Podstawowe pojęcia dotyczące baz danych

- Encja (relacja/tabela) - zbiór podobnych obiektów opisanych w jednolity sposób.
- Krotka (obiekt/rekord) – obiekt opisany wszystkimi atrybutami danej relacji.
- Związek (relacja/więz) – zależność występująca pomiędzy dwiema poszczególnymi tabelami.
- Atrybut – pojedyncza dana wchodząca w skład krotki określająca ją.
- Klucz główny – Taki zbiór atrybutów relacji, których kombinacje wartości jednoznacznie identyfikują każdą krotkę tej relacji a żaden podzbiór tego zbioru nie posiada tej własności. Jeśli zbiór ten jest jednoelementowy mówimy o kluczu prostym, jeśli jest wieloelementowy – o kluczu złożonym .
- Klucz obcy - atrybut lub zbiór atrybutów, wskazujący na klucz główny w innej relacji (tabeli). Klucz obcy to nic innego jak związek, relacja między dwoma tabelami. Definicja klucza obcego, pilnuje aby w tabeli powiązanej, w określonych atrybutach, znaleźć się mogły tylko takie wartości które istnieją w tabeli docelowej jako klucz główny. Klucz obcy może dotyczyć również tej samej tabeli.

Powiązania pomiędzy tabelami

Można mówić o trzech fundamentalnych związkach między relacjami . Dzięki nim można zapewnić integralność referencyjną danych i zamodelować pewną logikę struktury danych.

- Relacja jeden do jednego (1:1) – Każdy wiersz z tabeli A może mieć tylko jednego odpowiednika w tabeli B. Relacja 1:1 jest tworzona, jeżeli obie kolumny pokrewne są kluczami podstawowymi lub podlegają unikatowym ograniczeniom. Stosowana jest np. wtedy, gdy zbiór dodatkowych atrybutów jest określony tylko dla wąskiego podzbioru wierszy w tabeli podstawowej.
- Relacja jeden do wielu (1:N) – Każdy element z tabeli A może być powiązany z wieloma elementami z tabeli B. Pojedynczemu rekordowi z tabeli B odpowiada tylko jeden rekord z tabeli A.

- Relacja wiele do wielu (N:M) – Realizowana jest jako dwie relacje 1:N. Aby zamodelować relację N:M pomiędzy dwoma tabelami, potrzebna jest trzecia tabela zwana łącznikową. Brak wprowadzenia tabeli łącznikowej może skutkować redundancją danych.

1.2 Normalizacja

Normalizacja bazy danych to bezstratny proces organizowania danych w tabelach mający na celu zmniejszenie ilości danych składowanych w bazie oraz wyeliminowanie potencjalnych anomalii. Postać normalna - postać relacji w bazie danych, w której nie występuje redundancja (nadmiarowość), czyli powtarzanie się tych samych informacji. Doprowadzenia relacji do postaci normalnej nazywa się normalizacją (lub dekompozycją) bazy danych. Poniżej przedstawiony jest przykład normalizacji na podstawie realizowanej przez nas bazy danych.

Pierwsza postać normalna 1NF

Pierwsza postać normalna mówi o atomowości danych. Każde pole przechowuje jedną informację, dzięki czemu można dokonywać efektywnych zapytań. Wprowadza także istnienie klucza głównego. Kolejność wierszy może być dowolna, znaczenie danych nie zależy od kolejności wierszy. Tabela jest w pierwszej postaci normalnej jeśli: każdy wiersz przechowuje informację o pojedynczym obiekcie, nie zawiera kolekcji (powtarzających się grup informacji), wartości atrybutów są elementarne, posiada klucz główny. Typy przechowywanych danych powinny być najmniejsze z możliwych.

```
1. CREATE TABLE "SZKOLAJAZDY"."INSTRUKTORZY"  
2. (  
3.     "INSTRUKTOR_ID" NUMBER(19,0),  
4.     "CZY_ADMIN" NUMBER(19,0),  
5.     "DATA_DODANIA" DATE,  
6.     "EMAIL" VARCHAR2(20 CHAR),  
7.     "GODZ_ROZPOCZECIA" VARCHAR2(20 CHAR),  
8.     "GODZ_ZAKONCZENIA" VARCHAR2(20 CHAR),  
9.     "HASLO" VARCHAR2(20 CHAR),  
10.    "IMIE" VARCHAR2(20 CHAR),  
11.    "NAZWISKO" VARCHAR2(20 CHAR)  
12. )  
13. ALTER TABLE "SZKOLAJAZDY"."INSTRUKTORZY" ADD PRIMARY KEY ("INSTRUKTOR_ID")
```

Rysunek 1. Tabela INSTRUKTORZY spełniająca zasadę 1NF

Druga postać normalna 2NF

Druga postać normalna mówi o tym, że każda tabela powinna przechowywać dane dotyczące tylko konkretnej klasy obiektów. Należy wydzielić należy zbiór atrybutów, który jest zależny tylko od klucza głównego. Wszystkie atrybuty informacyjne (nie należące do klucza), muszą zawierać informacje o elementach tej konkretnej klasy (encji), a nie żadnej innej. Kolumny opisujące inne obiekty, powinny trafić do właściwych encji, w których te obiekty będziemy przechowywać.

Przykładem 2NF w naszej bazie danych może być to, że tabela instruktorzy nie przechowuje informacji o rezerwacjach, które są wykonane na konkretnego instruktora. Obiekt instruktor i rezerwacja znajdują się w różnych encjach powiązanych ze sobą poprzez klucz obcy (instruktor_id) wskazujący na klucz główny tabeli INSTRUKTORZY (instruktor_id).

```
1. CREATE TABLE "SZKOLAJAZDY"."REZERWACJE"  
2. (  
3.     "REZERWACJA_ID" NUMBER(19,0),  
4.     "DATA_DODANIA" DATE,  
5.     "DATA_REZERWACJI" DATE,  
6.     "GODZ_ROZPOCZECIA" VARCHAR2(20 CHAR),  
7.     "INSTRUKTOR_ID" NUMBER(19,0),  
8.     "KATEGORIA_ID" NUMBER(19,0),  
9.     "KURSANT_ID" NUMBER(19,0),  
10.    "USLUGA_ID" NUMBER(19,0)  
11. )  
12. ALTER TABLE "SZKOLAJAZDY"."REZERWACJE" ADD CONSTRAINT "FKOXVQVD63LTWGM9ATWSWQHPYL" FOREIGN KEY ("INSTRUKTOR_ID")  
13. REFERENCES "SZKOLAJAZDY"."INSTRUKTORZY" ("INSTRUKTOR_ID") ENABLE;
```

Rysunek 2. Tabela REZERWACJE przechowująca informację o odpowiadającym jej id instruktora

Trzecia postać normalna 3NF

Trzecia postać normalna mówi o tym, że kolumna informacyjna nie należąca do klucza nie zależy też od innej kolumny informacyjnej, nie należącej do klucza. Czyli każdy atrybut niekluczowy nie zależy funkcyjnie od innego atrybutu niekluczowego.

Przykładem 3NF w naszej bazie danych może być to, że w tabeli REZERWACJE pod uslugą_id kryją się nazwa i cena usługi. Cena usługi zależy bezpośrednio od jej nazwy, zatem gdybyśmy włączyli do tabeli te dwa dodatkowe atrybuty zasada 3NF nie byłaby spełniona. Rozwiązaniem problemu jest utworzenie osobnej tabeli USLUGI, która przechowuje informację o cenie i nazwie pojedynczej usługi.

```

1. CREATE TABLE "SZKOLAJAZDY"."USLUGI"
2. (   "USLUGA_ID" NUMBER(19,0),
3.     "CENA" NUMBER(19,0),
4.     "NAZWA" VARCHAR2(20 CHAR)
5. )

```

Rysunek 3. Tabela USŁUGI

2. Część praktyczna projektu

2.1 Przedstawienie problemu

Postanowiliśmy zrealizować prosty system **szkoły jazdy**. Projekt będzie realizowany w następujący sposób:

- Połączenie z bazą będzie realizowane przy pomocy aplikacji desktopowej. Dostępne w niej będą dwie formy początkowego logowania – jedna dla użytkownika i druga dla instruktora oraz jedna forma rejestracji – zakładanie nowego konta użytkownika.

Interfejs użytkownika

- Użytkownik może przeglądać historię rezerwacji (wraz z filtrowaniem rekordów), wyświetlać ich szczegóły (datę, czas trwania, w przypadku kursu – ilość zrealizowanych godzin, ilość brakujących godzin).
- Użytkownik może dodawać nowe rezerwacje do swojego konta.
- Użytkownik może sprawdzać saldo swojego konta.

Interfejs instruktora

- Instruktor może wyświetlać listę nadchodzących usług jakie zostały u niego zarezerwowane.
- Instruktor ma podgląd na swoich kursantów i ich obecne rezerwacje.
- Instruktor może wygenerować podsumowania swojej pracy za dany okres czasu tj. wygenerować plik ze swoimi danymi oraz rezerwacjami przypisanymi do swojego konta.

Interfejs administratora

- Administrator może wyświetlić listę rezerwacji i je w pełni modyfikować.
- Administrator może wyświetlić listę instruktorów, dodawać oraz edytować ich konta i nadawać prawo do kategorii.
- Administrator może wyświetlać i edytować konta użytkownika.
- Administrator może wyświetlać raport danego kursanta lub instruktora za dany okres czasu tj. wygenerować plik pokazujący dane osobowe oraz rezerwacje przypisane do danego konta.
- Administrator może wyświetlać listę usług i je modyfikować.

- Administrator może dodawać i edytować płatności za daną rezerwację, które odciążają saldo kursanta.

2.2 Wymagania systemu

- IntelliJ IDEA Ultimate 2018 – do uruchomienia aplikacji
- Oracle SQL Developer 11.2.0
- Oracle Database XE 11.2

2.3 Model danych ERD

Model ER (Entity Relationship Model) – opis teoretyczny relacyjnej bazy danych, który ma na celu opisanie jej za pomocą związków encji.

Diagram ERD (Entity Relationship Diagram) – graficzny odpowiednik modelu ER. Pozwala na zrozumienie struktury danych, przygotowania późniejszej strategii optymalizacji bazy, oraz stanowi podstawową dokumentację systemu przechowywania informacji. Przedstawia on obiekty, o których informacje są istotne w realizacji celów do których tworzona jest baza danych, ich atrybuty oraz związki pomiędzy nimi.

a) Identyfikacja zbioru encji wraz z ich atrybutami kluczowymi

W naszym systemie wyodrębniono encje oraz atrybuty jednoznacznie je identyfikujące.

Tabela 1. Encje i identyfikatory

| Encja | Klucz główny |
|-------------------------------|----------------------|
| <i>INSTRUKTORZY</i> | <i>INSTRUKTOR_ID</i> |
| <i>KATEGORIE</i> | <i>KATEGORIA_ID</i> |
| <i>KATEGORIE_INSTRUKTOROW</i> | <i>ID_WPISU</i> |
| <i>KURSANCY</i> | <i>KURSANT_ID</i> |
| <i>PLATNOSCI</i> | <i>PLATNOSC_ID</i> |
| <i>REZERWACJE</i> | <i>REZERWACJA_ID</i> |
| <i>USLUGI</i> | <i>USLUGA_ID</i> |

b) Identyfikacja bezpośrednich zależności między encjami

Tabela 2. Tabela krzyżowa - zależności bezpośrednie pomiędzy encjami

| | <i>INSTRUKTORZY</i> | <i>KATEGORIE</i> | <i>KATEGORIE_INSTRUKTOROW</i> | <i>KURSANCY</i> | <i>PLATNOSCI</i> | <i>REZERWACJE</i> | <i>USLUGI</i> |
|-------------------------------|---------------------|------------------|-------------------------------|-----------------|------------------|-------------------|---------------|
| <i>INSTRUKTORZY</i> | | | | | | | |
| <i>KATEGORIE</i> | | | | | | | |
| <i>KATEGORIE_INSTRUKTOROW</i> | X | X | | | | | |
| <i>KURSANCY</i> | | | | | | | |
| <i>PLATNOSCI</i> | | | | X | | | |
| <i>REZERWACJE</i> | X | X | | X | | | X |
| <i>USLUGI</i> | | | | | | | |

Tabela 3. Opis atrybutów tabeli "INSTRUKTORZY"

| INSTRUKTORZY | |
|-------------------------|---------------------------------------|
| Atrybut | Opis |
| <i>INSTRUKTOR_ID</i> | Uniwersalny identyfikator instruktora |
| <i>CZY_ADMIN</i> | Czy instruktor ma uprawnienia admina |
| <i>DATA_DODANIA</i> | Data dodania instruktora do systemu |
| <i>EMAIL</i> | Adres e-mail instruktora |
| <i>GODZ_ROZPOCZECIA</i> | Godzina rozpoczęcia pracy instruktora |
| <i>GODZ_ZAKONCZENIA</i> | Godzina zakończenia pracy instruktora |
| <i>HASLO</i> | Hasło logowania do systemu |
| <i>IMIE</i> | Imię instruktora |
| <i>NAZWISKO</i> | Nazwisko instruktora |

Tabela 4. Opis atrybutów tabeli "KATEGORIE"

| KATEGORIE | |
|---------------------|---|
| Atrybut | Opis |
| <i>KATEGORIA_ID</i> | Uniwersalny identyfikator kategorii prawa jazdy |
| <i>SYMBOL</i> | Symbol kategorii prawa jazdy (A,B itp.) |

Tabela 5. Opis atrybutów tabeli "KATEGORIE_INSTRUKTORÓW"

| KATEGORIE_INSTRUKTORÓW | |
|-------------------------------|--|
| Atrybut | Opis |
| <i>ID_WPISU</i> | Uniwersalny identyfikator wpisu, który wiąże instruktora z odpowiednią kategorią |
| <i>INSTRUCTOR_ID</i> | Uniwersalny identyfikator instruktora |
| <i>KATEGORIA_ID</i> | Uniwersalny identyfikator kategorii prawa jazdy |

Tabela 6. Opis atrybutów tabeli "KURSANCY"

| KURSANCY | |
|-------------------------|--------------------------------------|
| Atrybut | Opis |
| <i>KURSANT_ID</i> | Uniwersalny identyfikator kursanta |
| <i>DATA_REJESTRACJI</i> | Data rejestracji kursanta w systemie |
| <i>EMAIL</i> | Adres e-mail kursanta |
| <i>HASLO</i> | Hasło logowania do systemu |
| <i>IMIE</i> | Imię kursanta |
| <i>NAZWISKO</i> | Nazwisko kursanta |
| <i>PESEL</i> | Numer PESEL kursanta |
| <i>PKK</i> | Numer PKK kursanta |

Tabela 7. Opis atrybutów tabeli "PLATNOSCI"

| PLATNOSCI | |
|-----------------------|---|
| Atrybut | Opis |
| <i>PLATNOSC_ID</i> | Uniwersalny identyfikator płatności |
| <i>DATA_PLATNOSCI</i> | Data wykonania płatności |
| <i>KURSANT_ID</i> | Uniwersalny identyfikator kursanta dokonującego płatności |
| <i>KWOTA</i> | Kwota płatności |

Tabela 8. Opis atrybutów tabeli "REZERWACJE"

| REZERWACJE | |
|-------------------------|--|
| Atrybut | Opis |
| <i>REZERWACJA_ID</i> | Uniwersalny identyfikator rezerwacji |
| <i>DATA_DODANIA</i> | Data dodania rezerwacji do systemu |
| <i>DATA_REZERWACJI</i> | Data na którą rezerwacja ma być wykonana |
| <i>GODZ_ROZPOCZECIA</i> | Godzina rozpoczęcia rezerwacji |
| <i>GODZ_ZAKONCZENIA</i> | Godzina zakończenia rezerwacji |
| <i>INSTRUKTOR_ID</i> | Uniwersalny identyfikator instruktora obsługującego rezerwację |
| <i>KATEGORIA_ID</i> | Kategoria prawa jazdy, na którą jest zrobiona rezerwacja |
| <i>KURSANT_ID</i> | Uniwersalny identyfikator kursanta dokonującego rezerwacji |
| <i>USLUGA_ID</i> | Uniwersalny identyfikator usługi, która jest rezerwowana |

Tabela 9. Opis atrybutów tabeli "USLUGI"

| USLUGI | |
|------------------|----------------------------------|
| Atrybut | Opis |
| <i>USLUGA_ID</i> | Uniwersalny identyfikator usługi |
| <i>CENA</i> | Cena usługi |
| <i>NAZWA</i> | Nazwa usługi |

2.4 Schemat diagramu ERD

a) Opis aplikacji w której modelowano schemat

Schemat modelowano w aplikacji Oracle SQL Developer 11.2.0. SQL Developer przeznaczony jest pracy z bazą Oracle w wersji 9.2.0.1 oraz późniejszych. Ma on możliwość podłączenia do innych nie-Oraclovych baz danych, jednak dla ułatwienia bazę szkoły jazdy utworzono w Oracle Application Express.

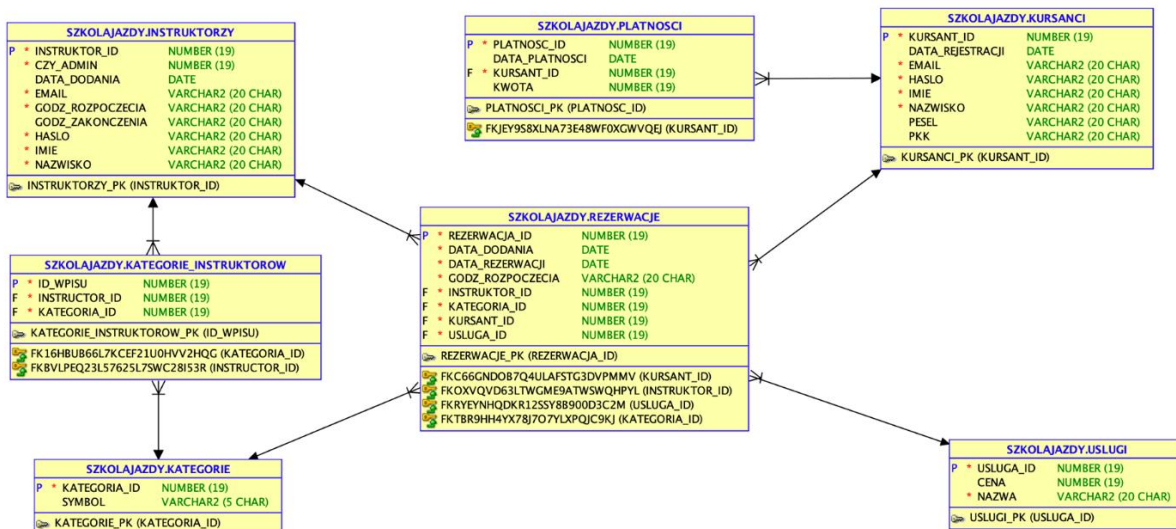
Edytor programisty wyposażony jest we wsparcie pisania kodu, polegające zarówno na podpowiedziach nazw obiektów, kolumn tabel itp., jak i na możliwości automatycznego wklejania szablonów struktur programowych, takich jak pętle, kursory itp.

SQL Developer umożliwia jednocześnie połączenie do wielu baz danych i pracy z nimi. Połączenia znajdują się w okienku *Connections* po lewej stronie. Umieszczając bazę w okienku połączeń mamy pogląd na strukturę bazy danych i łatwy dostęp do jednostki edytowanej w danej chwili. Okienko *Worksheet* przeznaczone jest do uruchamiania poleceń SQL. Po uruchomieniu napisanego przez nas skryptu otwiera się okienko *Script Output* w którym widoczny jest wynik naszego polecenia SQL. W menu *Run* znajdują się opcje krokowego uruchamiania programu oraz ustawiania pułapek.

SQL Developer jest darmowy, gwarantuje łatwość instalacji, uruchomienia i przenośność.

b) Prezentacja schematu ERD bazy danych

Schemat wygenerowano automatycznie, uprzednio implementując bazę w SQL Developerze.



Rysunek 4. Diagram ERD bazy "SZKOLAJAZDY"

2.5 Rozwiązanie problemu

a) System bazodanowy

Utworzenie bazy danych

Baza została utworzona przy pomocy Oracle Application Express. Po zalogowaniu się na ekranie widoczne jest okienko *Create Application Express Workspace*.

Create Application Express Workspace

Database User ☒ Create New ☐ Use Existing

* Database Username

* Application Express Username

* Password

* Confirm Password

Cancel Create Workspace

Rysunek 5. Tworzenie nowej bazy danych w Oracle Application Express

Następnie uruchomiono SQL Developer i utworzono nowe połączenie z bazą SZKOLAJAZDY, aby móc rozpocząć pracę na niej. W celu nawiązania połączenia należy kliknąć przycisk *Connect*.

New / Select Database Connection

| Connection Name | Connection Details |
|-----------------|-----------------------|
| HR | HR@//localhost:152... |
| SYSTEM | SYSTEM@//localhost... |
| SZKOLAJAZDY | SZKOLAJAZDY@//loc... |

Connection Name

Username

Password

☐ Save Password ☒ Connection Color

Oracle

Connection Type Role

Hostname

Port

☒ SID

☐ Service name

☐ OS Authentication ☐ Kerberos Authentication

Status :

Pomoc Save Clear Test Connect Anuluj

Rysunek 6. Nawiązywanie połączenia z bazą danych w SQL Developer

Po wykonaniu tej operacji w okienku *Connections* po lewej stronie widzimy nowe połączenie z bazą SZKOLAJAZDY.

Implementacja obiektów

Podstawowymi obiektami w bazie danych są tabele. Do utworzenia tabeli służy zapytanie **CREATE TABLE**. Składnia zapytania: **CREATE TABLE** nazwa tabeli (kolumna 1 typ_danych, kolumna 2 typ_danych, itp.), gdzie „kolumna...” definiuje nazwę kolumny (atrybutu), a typ_danych definiuje typ danych w danej kolumnie. Na typ danych możemy narzucić ograniczenia używając słowa kluczowego **CHECK** (..) i w nawiasie podając ograniczenia.

```
-----  
-- DDL for Table INSTRUKTORZY  
-----
```

```
CREATE TABLE "SZKOLAJAZDY"."INSTRUKTORZY"  
(  
  "INSTRUKTOR_ID" NUMBER(19,0),  
  "CZY_ADMIN" NUMBER(19,0),  
  "DATA_DODANIA" DATE,  
  "EMAIL" VARCHAR2(20 CHAR),  
  "GODZ_ROZPOCZECIA" VARCHAR2(20 CHAR),  
  "GODZ_ZAKONCZENIA" VARCHAR2(20 CHAR),  
  "HASLO" VARCHAR2(20 CHAR),  
  "IMIE" VARCHAR2(20 CHAR),  
  "NAZWISKO" VARCHAR2(20 CHAR)  
) SEGMENT CREATION IMMEDIATE  
PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255 NOCOMPRESS LOGGING  
STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645  
PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)  
TABLESPACE "USERS" ;
```

Rysunek 7. Tworzenie tabeli "INSTRUKTORZY"

```
-----  
-- DDL for Table KATEGORIE  
-----
```

```
CREATE TABLE "SZKOLAJAZDY"."KATEGORIE"  
(  
  "KATEGORIA_ID" NUMBER(19,0),  
  "SYMBOL" VARCHAR2(5 CHAR)  
) SEGMENT CREATION IMMEDIATE  
PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255 NOCOMPRESS LOGGING  
STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645  
PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)  
TABLESPACE "USERS" ;
```

Rysunek 8. Tworzenie tabeli "KATEGORIE"

```
-----  
-- DDL for Table KATEGORIE_INSTRUKTOROW  
-----
```

```
CREATE TABLE "SZKOLAJAZDY"."KATEGORIE_INSTRUKTOROW"  
(  
    "ID_WPISU" NUMBER(19,0),  
    "INSTRUCTOR_ID" NUMBER(19,0),  
    "KATEGORIA_ID" NUMBER(19,0)  
) SEGMENT CREATION IMMEDIATE  
PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255 NOCOMPRESS LOGGING  
STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645  
PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)  
TABLESPACE "USERS" ;
```

Rysunek 9. Tworzenie tabeli "KATEGORIE_INSTRUKTOROW"

```
-----  
-- DDL for Table KURSANCY  
-----
```

```
CREATE TABLE "SZKOLAJAZDY"."KURSANCY"  
(  
    "KURSANT_ID" NUMBER(19,0),  
    "DATA_REJESTRACJI" DATE,  
    "EMAIL" VARCHAR2(20 CHAR),  
    "HASLO" VARCHAR2(20 CHAR),  
    "IMIE" VARCHAR2(20 CHAR),  
    "NAZWISKO" VARCHAR2(20 CHAR),  
    "PESEL" VARCHAR2(20 CHAR),  
    "PKK" VARCHAR2(20 CHAR)  
) SEGMENT CREATION IMMEDIATE  
PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255 NOCOMPRESS LOGGING  
STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645  
PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)  
TABLESPACE "USERS" ;
```

Rysunek 10. Tworzenie tabeli "KURSANCY"

```
-----  
-- DDL for Table PLATNOSCI  
-----
```

```
CREATE TABLE "SZKOLAJAZDY"."PLATNOSCI"  
(  
  "PLATNOSC_ID" NUMBER(19,0),  
  "DATA_PLATNOSCI" DATE,  
  "KURSANT_ID" NUMBER(19,0),  
  "KWOTA" NUMBER(19,0)  
) SEGMENT CREATION IMMEDIATE  
PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255 NOCOMPRESS LOGGING  
STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645  
PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)  
TABLESPACE "USERS" ;
```

Rysunek 11. Tworzenie tabeli "PLATNOSCI"

```
-----  
-- DDL for Table REZERWACJE  
-----
```

```
CREATE TABLE "SZKOLAJAZDY"."REZERWACJE"  
(  
  "REZERWACJA_ID" NUMBER(19,0),  
  "DATA_DODANIA" DATE,  
  "DATA_REZERWACJI" DATE,  
  "GODZ_ROZPOCZECIA" VARCHAR2(20 CHAR),  
  "INSTRUKTOR_ID" NUMBER(19,0),  
  "KATEGORIA_ID" NUMBER(19,0),  
  "KURSANT_ID" NUMBER(19,0),  
  "USLUGA_ID" NUMBER(19,0)  
) SEGMENT CREATION IMMEDIATE  
PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255 NOCOMPRESS LOGGING  
STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645  
PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)  
TABLESPACE "USERS" ;
```

Rysunek 12. Tworzenie tabeli "REZERWACJE"

```
-----  
-- DDL for Table USLUGI  
-----
```

```
CREATE TABLE "SZKOLAJAZDY"."USLUGI"  
(  
  "USLUGA_ID" NUMBER(19,0),  
  "CENA" NUMBER(19,0),  
  "NAZWA" VARCHAR2(20 CHAR)  
) SEGMENT CREATION IMMEDIATE  
PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255 NOCOMPRESS LOGGING  
STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645  
PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)  
TABLESPACE "USERS" ;
```

Rysunek 13. Tworzenie tabeli "USLUGI"

Wprowadzenie danych

Zapytanie **INSERT** służy do dodania do tabeli nowych rekordów zawierających w odpowiednich kolumnach podane wartości. Składnia zapytania: **INSERT INTO** <nazwa tabeli> **VALUES** ('wartość1','wartość2', itp.), gdzie **INSERT INTO** oznacza początek operacji wstawiania, w nawiasie za słowem kluczowym **VALUES** definiujemy odpowiednie wartości w takiej kolejności, w jakiej występują one w tabeli.

Wprowadzenie danych instruktorów

| INSERT INTO INSTRUKTORZY | INSERT INTO INSTRUKTORZY | INSERT INTO INSTRUKTORZY |
|---|---|--|
| VALUES ('105', '1', '19/05/28', 'instruktor1@wp.pl', '6', '15', 'instruktor1', 'Jan', 'Nowak' | VALUES ((SELECT MAX(instruktor_id)+1 FROM INSTRUKTORZY), '1', '19/06/08', 'instruktor2@wp.pl', '8', '16', 'instruktor2', 'Tomasz', 'Cudny' | VALUES ((SELECT MAX(instruktor_id)+1 FROM INSTRUKTORZY), '0', '19/06/03', 'instruktor3@wp.pl', '10', '19', 'instruktor3', 'Jacek', 'Szkłanka' |
|); |); |); |

Wprowadzenie danych kursantów

```
INSERT INTO KURSANCY
VALUES(
(SELECT COUNT(kursant_id)+1 FROM
KURSANCY),
'19/06/08',
'emiliaaug@gmail.com',
'emilia123',
'Emilia',
'Augustyn',
'99010512345',
'11111111111111111111'
);

INSERT INTO KURSANCY
VALUES(
(SELECT COUNT(kursant_id)+1 FROM
KURSANCY),
'19/05/28',
'matsliwka@gmail.com',
'mateusz123',
'Mateusz',
'Sliwka',
'98070112345',
'22222222222222222222'
);
```

W podobny sposób wprowadzono jeszcze trzech kursantów aby tabela **KURSANCY** wyglądała tak:

| KURSANT_ID | DATA_REJESTRACJI | EMAIL | HASLO | IMIE | NAZWISKO | PESEL | PKK |
|------------|------------------|--------------------|------------|---------|-------------|-------------|----------------------|
| 1 | 4 19/05/31 | marekg@gmail.com | marek123 | Marek | Piórnik | 98010312345 | 44444444444444444444 |
| 2 | 1 19/06/08 | emiliaa@gmail.com | emilial23 | Emilia | Augustyn | 99010512345 | 11111111111111111111 |
| 3 | 2 19/05/28 | mateuszs@gmail.com | mateusz123 | Mateusz | Sliwka | 98070112345 | 22222222222222222222 |
| 4 | 5 19/04/29 | monikas@gmail.com | monikal23 | Monika | Sobierajska | 97060512345 | 55555555555555555555 |
| 5 | 3 19/06/01 | joannak@gmail.com | joanna123 | Joanna | Komorniczak | 98040512345 | 33333333333333333333 |

Wprowadzanie danych kategorii

```
REM INSERTING INTO SZKOLAJAZDY.KATEGORIE
SET DEFINE OFF;

INSERT INTO SZKOLAJAZDY.KATEGORIE (KATEGORIA_ID,SYMBOL) VALUES ('200','A');
INSERT INTO SZKOLAJAZDY.KATEGORIE (KATEGORIA_ID,SYMBOL) VALUES ('201','B');
INSERT INTO SZKOLAJAZDY.KATEGORIE (KATEGORIA_ID,SYMBOL) VALUES ('203','C');
INSERT INTO SZKOLAJAZDY.KATEGORIE (KATEGORIA_ID,SYMBOL) VALUES ('204','D');
INSERT INTO SZKOLAJAZDY.KATEGORIE (KATEGORIA_ID,SYMBOL) VALUES ('205','T');
```


Przypisywanie odpowiednich kategorii, które obsługuje instruktor

W bazie każdy instruktor obsługuje dwie różne kategorie praw jazdy:

```
INSERT INTO SZKOLAJAZDY.KATEGORIE_INSTRUKTOROW (ID_WPISU, INSTRUCTOR_ID, KATEGORIA_ID) VALUES ('113', '105', '204');
INSERT INTO SZKOLAJAZDY.KATEGORIE_INSTRUKTOROW (ID_WPISU, INSTRUCTOR_ID, KATEGORIA_ID) VALUES ('112', '105', '200');
INSERT INTO SZKOLAJAZDY.KATEGORIE_INSTRUKTOROW (ID_WPISU, INSTRUCTOR_ID, KATEGORIA_ID) VALUES ('111', '106', '203');
INSERT INTO SZKOLAJAZDY.KATEGORIE_INSTRUKTOROW (ID_WPISU, INSTRUCTOR_ID, KATEGORIA_ID) VALUES ('110', '106', '204');
INSERT INTO SZKOLAJAZDY.KATEGORIE_INSTRUKTOROW (ID_WPISU, INSTRUCTOR_ID, KATEGORIA_ID) VALUES ('109', '107', '200');
INSERT INTO SZKOLAJAZDY.KATEGORIE_INSTRUKTOROW (ID_WPISU, INSTRUCTOR_ID, KATEGORIA_ID) VALUES ('108', '107', '201');
```

Wprowadzenie danych usług

```
REM INSERTING INTO SZKOLAJAZDY.USLUGI
SET DEFINE OFF;
INSERT INTO SZKOLAJAZDY.USLUGI (USLUGA_ID, CENA, NAZWA) VALUES ((SELECT COUNT(usluga_id)+1 FROM USLUGI), '900', 'X');
INSERT INTO SZKOLAJAZDY.USLUGI (USLUGA_ID, CENA, NAZWA) VALUES ((SELECT COUNT(usluga_id)+1 FROM USLUGI), '1000', 'Y');
INSERT INTO SZKOLAJAZDY.USLUGI (USLUGA_ID, CENA, NAZWA) VALUES ((SELECT COUNT(usluga_id)+1 FROM USLUGI), '750', 'Z');
INSERT INTO SZKOLAJAZDY.USLUGI (USLUGA_ID, CENA, NAZWA) VALUES ((SELECT COUNT(usluga_id)+1 FROM USLUGI), '860', 'Q');
```

Zdefiniowanie w języku SQL poleceń dla realizacji typowych operacji

Wstawianie – INSERT

- Wstawienie nowej rezerwacji dla kursanta o identyfikatorze równym 1

```
INSERT INTO REZERWACJE
VALUES (
'1',
'19/06/09',
'19/06/11',
'9',
'106',
'203',
'1',
'1'
);
```

- Wstawianie nowego instruktora, który będzie obsługiwał kategorię prawa jazdy A i T oraz pracował o godzinę dłużej niż najpóźniej pracujący instruktor

```

INSERT INTO INSTRUKTORZY
VALUES (
(SELECT MAX(INSTRUKTOR_ID)+1 FROM INSTRUKTORZY),
'0',
'19/06/06',
'instruktor4@wp.pl',
'12',
(SELECT MAX(GODZ_ZAKONCZENIA)+1 FROM INSTRUKTORZY),
'instruktor4',
'Marian',
'Flamaster'
);

INSERT INTO KATEGORIE_INSTRUKTOROW
VALUES (
'107',
(SELECT INSTRUKTOR_ID FROM INSTRUKTORZY
WHERE IMIE='Marian'),
(SELECT KATEGORIA_ID FROM KATEGORIE
WHERE SYMBOL='A')
);

INSERT INTO KATEGORIE_INSTRUKTOROW
VALUES (
'106',
(SELECT INSTRUKTOR_ID FROM INSTRUKTORZY
WHERE IMIE='Marian'),
(SELECT KATEGORIA_ID FROM KATEGORIE
WHERE SYMBOL='T')
);

```

- Wstawienie nowej usługi 'W', która będzie najtańszą spośród wszystkich usług

```

INSERT INTO USLUGI
VALUES (
(SELECT MAX(USLUGA_ID)+1 FROM USLUGI),
(SELECT MIN(CENA)-100 FROM USLUGI),
'W'
);

```

- Wstawienie nowej płatności za kursanta, który ostatnio zarezerwował usługę 'X'

```

INSERT INTO PLATNOSCI
VALUES (
(SELECT COUNT(PLATNOSC_ID)+1 FROM PLATNOSCI),
'19/06/09',
(SELECT KURSANT_ID FROM REZERWACJE
JOIN USLUGI ON REZERWACJE.USLUGA_ID=USLUGI.USLUGA_ID
WHERE DATA_REZERWACJI = (SELECT MIN(DATA_REZERWACJI) FROM REZERWACJE)
AND NAZWA='X'),
(SELECT CENA FROM USLUGI WHERE NAZWA='X')
);

```

Usuwanie – DELETE

- Usunięcie wszystkich rezerwacji na kategorię 'A'

```
DELETE FROM REZERWACJE
WHERE KATEGORIA_ID = (SELECT KATEGORIA_ID FROM KATEGORIE
                      WHERE SYMBOL='A');
```

- Usunięcie najpóźniej dodanej płatności

```
DELETE FROM PLATNOSCI
WHERE DATA_PLATNOSCI = (SELECT MAX(DATA_PLATNOSCI) FROM PLATNOSCI);
```

- Usunięcie obsługi kategorii 'A' i 'B' przez instruktorów, którzy pracują po godzinie 16:00

```
DELETE FROM KATEGORIE_INSTRUKTOROW
WHERE INSTRUCTOR_ID IN (SELECT INSTRUCTOR_ID FROM INSTRUKTORZY
                        WHERE GODZ_ZAKONCZENIA>'16');
```

- Usunięcie rezerwacji kursantów, którzy urodzili się w 99' roku

```
DELETE FROM REZERWACJE
WHERE KURSANT_ID = (SELECT KURSANT_ID FROM KURSANCY
                   WHERE PESEL LIKE '99%');
```

- Usunięcie usługi, która była zarezerwowana tydzień temu

```
DELETE FROM USLUGI
WHERE USLUGA_ID = (SELECT USLUGA_ID FROM REZERWACJE
                   WHERE DATA_DODANIA = (SELECT CURRENT_DATE-7 FROM DUAL)
                   );
```

Modyfikacja – ALTER/UPDATE

Przypisanie klucza głównego do atrybutu i modyfikacja odpowiednich kolumn, aby nie mogły przyjąć wartości NULL

- Dla tabeli **INSTRUKTORZY**:

```
-----  
-- Constraints for Table INSTRUKTORZY  
-----
```

```
ALTER TABLE "SZKOLAJAZDY"."INSTRUKTORZY" ADD PRIMARY KEY ("INSTRUKTOR_ID")  
USING INDEX PCTFREE 10 INITRANS 2 MAXTRANS 255  
STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645  
PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)  
TABLESPACE "USERS" ENABLE;  
ALTER TABLE "SZKOLAJAZDY"."INSTRUKTORZY" MODIFY ("NAZWISKO" NOT NULL ENABLE);  
ALTER TABLE "SZKOLAJAZDY"."INSTRUKTORZY" MODIFY ("IMIE" NOT NULL ENABLE);  
ALTER TABLE "SZKOLAJAZDY"."INSTRUKTORZY" MODIFY ("HASLO" NOT NULL ENABLE);  
ALTER TABLE "SZKOLAJAZDY"."INSTRUKTORZY" MODIFY ("GODZ_ROZPOCZECIA" NOT NULL ENABLE);  
ALTER TABLE "SZKOLAJAZDY"."INSTRUKTORZY" MODIFY ("EMAIL" NOT NULL ENABLE);  
ALTER TABLE "SZKOLAJAZDY"."INSTRUKTORZY" MODIFY ("CZY_ADMIN" NOT NULL ENABLE);  
ALTER TABLE "SZKOLAJAZDY"."INSTRUKTORZY" MODIFY ("INSTRUKTOR_ID" NOT NULL ENABLE);
```

- Dla tabeli **KURSANCY**:

```
-----  
-- Constraints for Table KURSANCY  
-----
```

```
ALTER TABLE "SZKOLAJAZDY"."KURSANCY" ADD PRIMARY KEY ("KURSANT_ID")  
USING INDEX PCTFREE 10 INITRANS 2 MAXTRANS 255  
STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645  
PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)  
TABLESPACE "USERS" ENABLE;  
ALTER TABLE "SZKOLAJAZDY"."KURSANCY" MODIFY ("NAZWISKO" NOT NULL ENABLE);  
ALTER TABLE "SZKOLAJAZDY"."KURSANCY" MODIFY ("IMIE" NOT NULL ENABLE);  
ALTER TABLE "SZKOLAJAZDY"."KURSANCY" MODIFY ("HASLO" NOT NULL ENABLE);  
ALTER TABLE "SZKOLAJAZDY"."KURSANCY" MODIFY ("EMAIL" NOT NULL ENABLE);  
ALTER TABLE "SZKOLAJAZDY"."KURSANCY" MODIFY ("KURSANT_ID" NOT NULL ENABLE);
```

- Dla tabeli **REZERWACJE**:

```
-----  
-- Constraints for Table REZERWACJE  
-----
```

```
ALTER TABLE "SZKOLAJAZDY"."REZERWACJE" ADD PRIMARY KEY ("REZERWACJA_ID")  
USING INDEX PCTFREE 10 INITRANS 2 MAXTRANS 255  
STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645  
PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)  
TABLESPACE "USERS" ENABLE;  
ALTER TABLE "SZKOLAJAZDY"."REZERWACJE" MODIFY ("USLUGA_ID" NOT NULL ENABLE);  
ALTER TABLE "SZKOLAJAZDY"."REZERWACJE" MODIFY ("KURSANT_ID" NOT NULL ENABLE);  
ALTER TABLE "SZKOLAJAZDY"."REZERWACJE" MODIFY ("KATEGORIA_ID" NOT NULL ENABLE);  
ALTER TABLE "SZKOLAJAZDY"."REZERWACJE" MODIFY ("INSTRUKTOR_ID" NOT NULL ENABLE);  
ALTER TABLE "SZKOLAJAZDY"."REZERWACJE" MODIFY ("GODZ_ROZPOCZECIA" NOT NULL ENABLE);  
ALTER TABLE "SZKOLAJAZDY"."REZERWACJE" MODIFY ("DATA_REZERWACJI" NOT NULL ENABLE);  
ALTER TABLE "SZKOLAJAZDY"."REZERWACJE" MODIFY ("DATA_DODANIA" NOT NULL ENABLE);  
ALTER TABLE "SZKOLAJAZDY"."REZERWACJE" MODIFY ("REZERWACJA_ID" NOT NULL ENABLE);
```

- Dla tabeli **USLUGI**:

```
-----  
-- Constraints for Table USLUGI  
-----
```

```
ALTER TABLE "SZKOLAJAZDY"."USLUGI" ADD PRIMARY KEY ("USLUGA_ID")  
USING INDEX PCTFREE 10 INITRANS 2 MAXTRANS 255  
STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645  
PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)  
TABLESPACE "USERS" ENABLE;  
ALTER TABLE "SZKOLAJAZDY"."USLUGI" MODIFY ("NAZWA" NOT NULL ENABLE);  
ALTER TABLE "SZKOLAJAZDY"."USLUGI" MODIFY ("USLUGA_ID" NOT NULL ENABLE);
```

- Dla tabeli **KATEGORIE**:

```
-----  
-- Constraints for Table KATEGORIE  
-----
```

```
ALTER TABLE "SZKOLAJAZDY"."KATEGORIE" ADD PRIMARY KEY ("KATEGORIA_ID")  
USING INDEX PCTFREE 10 INITRANS 2 MAXTRANS 255  
STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645  
PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)  
TABLESPACE "USERS" ENABLE;  
ALTER TABLE "SZKOLAJAZDY"."KATEGORIE" MODIFY ("KATEGORIA_ID" NOT NULL ENABLE);
```

- Dla tabeli **KATEGORIE_INSTRUKTOROW**:

```
-----
-- Constraints for Table KATEGORIE_INSTRUKTOROW
-----
```

```
ALTER TABLE "SZKOLAJAZDY"."KATEGORIE_INSTRUKTOROW" ADD PRIMARY KEY ("ID_WPISU")
USING INDEX PCTFREE 10 INITRANS 2 MAXTRANS 255
STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)
TABLESPACE "USERS" ENABLE;

ALTER TABLE "SZKOLAJAZDY"."KATEGORIE_INSTRUKTOROW" MODIFY ("KATEGORIA_ID" NOT NULL ENABLE);
ALTER TABLE "SZKOLAJAZDY"."KATEGORIE_INSTRUKTOROW" MODIFY ("INSTRUCTOR_ID" NOT NULL ENABLE);
ALTER TABLE "SZKOLAJAZDY"."KATEGORIE_INSTRUKTOROW" MODIFY ("ID_WPISU" NOT NULL ENABLE);
```

- Dla tabeli **PLATNOSCI**:

```
-----
-- Constraints for Table PLATNOSCI
-----
```

```
ALTER TABLE "SZKOLAJAZDY"."PLATNOSCI" ADD PRIMARY KEY ("PLATNOSC_ID")
USING INDEX PCTFREE 10 INITRANS 2 MAXTRANS 255
STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)
TABLESPACE "USERS" ENABLE;

ALTER TABLE "SZKOLAJAZDY"."PLATNOSCI" MODIFY ("KURSANT_ID" NOT NULL ENABLE);
ALTER TABLE "SZKOLAJAZDY"."PLATNOSCI" MODIFY ("PLATNOSC_ID" NOT NULL ENABLE);
```

Ustawienie kluczy obcych

- W tabeli **KATEGORIE_INSTRUKTOROW**:

```
-----
-- Ref Constraints for Table KATEGORIE_INSTRUKTOROW
-----
```

```
ALTER TABLE "SZKOLAJAZDY"."KATEGORIE_INSTRUKTOROW" ADD CONSTRAINT "FK16HBUB66L7KCEF21U0HV2H2HQG" FOREIGN KEY ("KATEGORIA_ID")
REFERENCES "SZKOLAJAZDY"."KATEGORIE" ("KATEGORIA_ID") ENABLE;
ALTER TABLE "SZKOLAJAZDY"."KATEGORIE_INSTRUKTOROW" ADD CONSTRAINT "FKBVLPEQ23L57625L7SWC28I53R" FOREIGN KEY ("INSTRUCTOR_ID")
REFERENCES "SZKOLAJAZDY"."INSTRUKTORZY" ("INSTRUKTOR_ID") ENABLE;
```

- W tabeli **PLATNOSCI**:

```
-----
-- Ref Constraints for Table PLATNOSCI
-----
```

```
ALTER TABLE "SZKOLAJAZDY"."PLATNOSCI" ADD CONSTRAINT "FKJEY9S8XLNA73E48WF0XGWVQEJ" FOREIGN KEY ("KURSANT_ID")
REFERENCES "SZKOLAJAZDY"."KURSANCY" ("KURSANT_ID") ENABLE;
```

- W tabeli **REZERWACJE**:

```
-----
-- Ref Constraints for Table REZERWACJE
-----
```

```
ALTER TABLE "SZKOLAJAZDY"."REZERWACJE" ADD CONSTRAINT "FKC66GND0B7Q4ULAFSTG3DVPMMV" FOREIGN KEY ("KURSANT_ID")
REFERENCES "SZKOLAJAZDY"."KURSANCY" ("KURSANT_ID") ENABLE;
ALTER TABLE "SZKOLAJAZDY"."REZERWACJE" ADD CONSTRAINT "FKOXVQVD63LTWGM9ATWSWQHPYL" FOREIGN KEY ("INSTRUKTOR_ID")
REFERENCES "SZKOLAJAZDY"."INSTRUKTORZY" ("INSTRUKTOR_ID") ENABLE;
ALTER TABLE "SZKOLAJAZDY"."REZERWACJE" ADD CONSTRAINT "FKRYEYNHQDKR12SSY8B900D3C2M" FOREIGN KEY ("USLUGA_ID")
REFERENCES "SZKOLAJAZDY"."USLUGI" ("USLUGA_ID") ENABLE;
ALTER TABLE "SZKOLAJAZDY"."REZERWACJE" ADD CONSTRAINT "FKTBR9HH4YX78J7O7YLPQJC9KJ" FOREIGN KEY ("KATEGORIA_ID")
REFERENCES "SZKOLAJAZDY"."KATEGORIE" ("KATEGORIA_ID") ENABLE;
```

Modyfikowanie wartości w tabelach

- Przeceniecie usługi 'X' o 20%

```
UPDATE USLUGI
SET CENA = CENA*0.8
WHERE NAZWA = 'X';
```

- Zabranie wszystkim instruktorom uprawnień admina

```
UPDATE INSTRUKTORZY
SET CZY_ADMIN='0';
```

- Zmiana kwoty płatności na 1500,- kursantom, którzy zarezerwowali kurs na kategorię 'A' u instruktora Mariana

```
UPDATE PLATNOSCI
SET KWOTA='1500'
WHERE KURSANT_ID = (SELECT KURSANT_ID FROM REZERWACJE
                    JOIN KATEGORIE ON REZERWACJE.KATEGORIA_ID=KATEGORIE.KATEGORIA_ID
                    JOIN INSTRUKTORZY ON REZERWACJE.INSTRUKTOR_ID=INSTRUKTORZY.INSTRUKTOR_ID
                    WHERE KATEGORIE.SYMBOL='A' AND INSTRUKTORZY.IMIE='Marian');
```

- Zmiana godziny rozpoczęcia rezerwacji na 10:00 kursanta, który zarezerwował dziś usługę Y u instruktora Jacka

```
UPDATE REZERWACJE
SET GODZ_ROZPOCZECIA = '10'
WHERE DATA_DODANIA = (SELECT CURRENT_DATE FROM DUAL)
AND USLUGA_ID = (SELECT USLUGA_ID FROM USLUGI WHERE NAZWA = 'Y')
AND INSTRUKTOR_ID = (SELECT INSTRUKTOR_ID FROM INSTRUKTORZY WHERE IMIE='Jacek');
```

- Zmiana kategorii 'C' na 'T' instruktorom, którzy obsługiwali kategorię 'C'

```
UPDATE KATEGORIE_INSTRUKTOROW
SET KATEGORIA_ID = (SELECT KATEGORIA_ID FROM KATEGORIE WHERE SYMBOL='T')
WHERE KATEGORIA_ID= (SELECT KATEGORIA_ID FROM KATEGORIE WHERE SYMBOL='C');
```

- Zmiana instruktora w rezerwacji kursanta na instruktora o podanym nazwisku

```
UPDATE REZERWACJE
SET INSTRUKTOR_ID = (SELECT INSTRUKTOR_ID FROM INSTRUKTORZY
                     WHERE NAZWISKO='<strong>snazwisko')
WHERE KURSANT_ID = '122';
```

Selekcja/prezentacja – SELECT

- Wybranie wszystkich rekordów z ich atrybutami z każdej tabeli w bazie, '*' oznacza wybranie wszystkich:

```
SELECT * FROM INSTRUKTORZY;
SELECT * FROM KATEGORIE;
SELECT * FROM KATEGORIE_INSTRUKTOROW;
SELECT * FROM KURSANCY;
SELECT * FROM PLATNOSCI;
SELECT * FROM REZERWACJE;
SELECT * FROM USLUGI;
```

- Wybranie kursantów, którzy zrobili rezerwację na usługę o nazwie 'X'

```
SELECT KURSANCY.KURSANT_ID, IMIE, NAZWISKO, USLUGI.NAZWA FROM KURSANCY
JOIN REZERWACJE ON KURSANCY.KURSANT_ID=REZERWACJE.KURSANT_ID
JOIN USLUGI ON REZERWACJE.USLUGA_ID=USLUGI.USLUGA_ID
WHERE USLUGI.NAZWA='X';
```


- Wybranie instruktora, na którego zrobione jest najwięcej rezerwacji

```
SELECT INSTRUKTORZY.INSTRUKTOR_ID, INSTRUKTORZY.IMIE, INSTRUKTORZY.NAZWISKO
FROM INSTRUKTORZY
JOIN REZERWACJE ON INSTRUKTORZY.INSTRUKTOR_ID=REZERWACJE.INSTRUKTOR_ID
WHERE INSTRUKTORZY.INSTRUKTOR_ID = (SELECT INSTRUKTOR_ID FROM REZERWACJE
                                     HAVING COUNT(REZERWACJA_ID) = (SELECT MAX(COUNT(REZERWACJA_ID))
                                                                     FROM REZERWACJE GROUP BY REZERWACJA_ID)
                                     GROUP BY INSTRUKTOR_ID, REZERWACJA_ID
                                    );
```

- Wybranie instruktora, który najpóźniej kończy pracę

```
SELECT IMIE, NAZWISKO, GODZ_ZAKONCZENIA FROM INSTRUKTORZY
WHERE GODZ_ZAKONCZENIA = (SELECT MAX(GODZ_ZAKONCZENIA)
                          FROM INSTRUKTORZY);
```

- Wybranie wszystkich instruktorów którzy obsługują kategorie prawa jazdy 'B'

```
SELECT INSTRUKTOR_ID, IMIE, NAZWISKO FROM INSTRUKTORZY
JOIN KATEGORIE_INSTRUKTOROW ON INSTRUKTORZY.INSTRUKTOR_ID=KATEGORIE_INSTRUKTOROW.INSTRUKTOR_ID
JOIN KATEGORIE ON KATEGORIE_INSTRUKTOROW.KATEGORIA_ID=KATEGORIE.KATEGORIA_ID
WHERE KATEGORIE.SYMBOL='B';
```

- Wybranie kursanta, który ostatnio dokonał płatności

```
SELECT KURSANTCI.KURSANT_ID, IMIE, NAZWISKO FROM KURSANTCI
JOIN PLATNOSCI ON KURSANTCI.KURSANT_ID=PLATNOSCI.KURSANT_ID
WHERE DATA_PLATNOSCI = (SELECT MIN(DATA_PLATNOSCI) FROM PLATNOSCI);
```

- Sprawdzenie, czy można dodać nową rezerwację do instruktora w podanym terminie

```
-- Sprawdzamy czy sa rezerwacje na ta godzine --
```

```
SELECT * FROM REZERWACJE
WHERE GODZ_ROZPOCZECIA = '9';
```

```
-- Sprawdzamy czy godzina sie mieści w przedziale pracy instruktora --
```

```
SELECT * FROM INSTRUKTORZY
WHERE '9' BETWEEN CAST(GODZ_ROZPOCZECIA AS INT) AND CAST(GODZ_ZAKONCZENIA AS INT)
AND INSTRUKTOR_ID='106';
```

```
-- Jeżeli pierwsze zapytanie nie zwróci nic i drugie zapytanie zwróci wynik to
-- rezerwację dodajemy --
```

b) Aplikacja

Charakterystyka

Aplikacja została stworzona w języku Java z wykorzystaniem framework'a Hibernate. Hibernate zapewnia translację danych pomiędzy relacyjną bazą danych a światem obiekowym. Opiera się na wykorzystaniu opisu struktury danych za pomocą języka XML, dzięki czemu można rzutować obiekty, stosowane w obiektowych językach programowania, takich jak Java bezpośrednio na istniejące tabele bazy danych.

Aplikacja jest okienkowa, posiada ona trzy interfejsy: użytkownika, instruktora i administratora. W zależności od uprawnień mogą oni wykonywać różne operacje, dokładnie zostało to opisane w podpunkcie *Przedstawienie problemu*. (Ctrl + kliknięcie śledzi łączy).

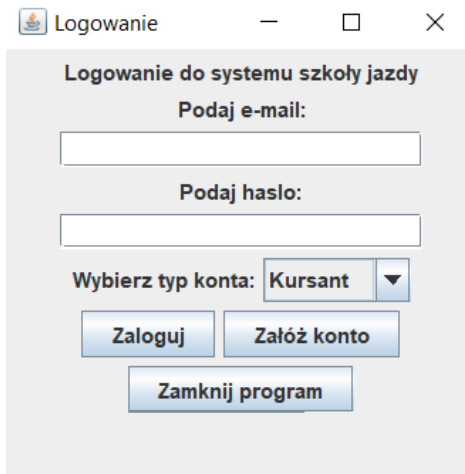
Implementacja

Aplikacja została zaimplementowana w środowisku IntelliJ IDEA. Podłączono do niej bazę, którą stworzono wcześniej.

Opis formularzy i operacji – najważniejsze wybrane operacje

1. Panel logowania

Na początku po uruchomieniu się aplikacji widzimy okienko logowania. Kluczowa jest tutaj funkcja *actionPerformed*, która za parametr przyjmuje wybraną przez nas opcję: logowania, założenia nowego konta lub zamknięcia programu.



The screenshot shows a Java Swing window titled "Logowanie" (Login). The window has a standard title bar with a minimize button, a maximize button (disabled), and a close button. The main content area has a light gray background and contains the following elements:

- A title "Logowanie do systemu szkoły jazdy" (Login to the driving school system) in bold black text.
- A label "Podaj e-mail:" followed by a text input field.
- A label "Podaj hasło:" followed by a text input field.
- A label "Wybierz typ konta:" followed by a dropdown menu currently showing "Kursant" (Student) with a downward arrow.
- Three buttons at the bottom: "Zaloguj" (Login), "Założ konto" (Create account), and "Zamknij program" (Close program).

Rysunek 14. Okienko logowania

```

@Override
public void actionPerformed(ActionEvent e) {
    Object source = e.getSource();
    if (source == register) {
        new RegisterFrame();
    }
    if (source == zaloguj) {
        if (typBox.getSelectedItem().toString() == "Kursant") {
            KursanciEntity kursant = new KursanciController().login(login.getText(), haslo.getText());
            if (kursant == null) {
                niezalogowano();
            } else {
                zalogowano();
                new UserFrame(kursant);
            }
        } else if (typBox.getSelectedItem().toString() == "Instruktor") {
            InstruktorzyEntity instruktor = new InstruktorzyController().login(login.getText(), haslo.getText());
            if (instruktor == null) {
                niezalogowano();
            } else {
                zalogowano();
                new LoggedInFrame(instruktor);
            }
        }
    }
    else if (source == close) {
        int decyzja = JOptionPane.showConfirmDialog( parentComponent: this, message: "Czy na pewno chcesz zamknąć program?",
            title: "Potwierdź zamykanie", JOptionPane.YES_NO_OPTION);
        if (decyzja == 0) {
            Window win = SwingUtilities.getWindowAncestor( C this);
            ((Window) win).dispose();
        }
    }
}
}

```

Rysunek 15. Kod programu

Logowanie do panelu kursanta

Jeżeli wybraliśmy opcję kursant, wykonuje się funkcja *login*:

```

public KursanciEntity login(String email, String haslo) {
    List<KursanciEntity> result = (List<KursanciEntity>) entityManager.createQuery( S: "SELECT u FROM KursanciEntity u " +
        "WHERE u.email=:email AND u.haslo=:haslo")
        .setParameter( S: "email", email).setParameter( S: "haslo", haslo).getResultList();
    if (result.isEmpty())
        return null;
    return result.get(0);
}

```

Wykonuje ona zapytanie do bazy, które zwraca czy taki kursant znajduje się w bazie. Jeśli nie, funkcja zwraca *null*.

Logowanie do panelu instruktora

Podobnie wykonuje się funkcja *login*:

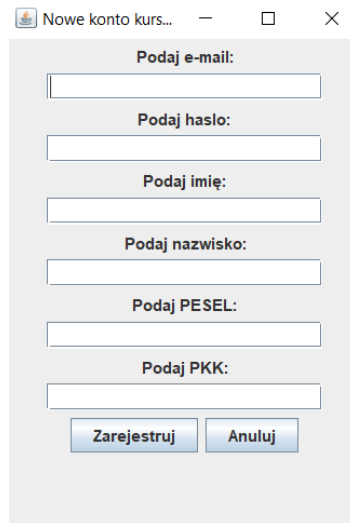
```

public InstruktorzyEntity login(String email, String haslo) {
    List<InstruktorzyEntity> result = entityManager.createQuery( S: "SELECT u FROM InstruktorzyEntity u " +
        "WHERE u.email=:email AND u.haslo=:haslo")
        .setParameter( S: "email", email).setParameter( S: "haslo", haslo).getResultList();
    if (result.isEmpty())
        return null;
    return result.get(0);
}

```

Wykonuje ona analogiczne do powyższego zapytanie do bazy, czy taki instruktor się w niej znajduje.

Rejestracja



Rysunek 16. Okienko rejestracji

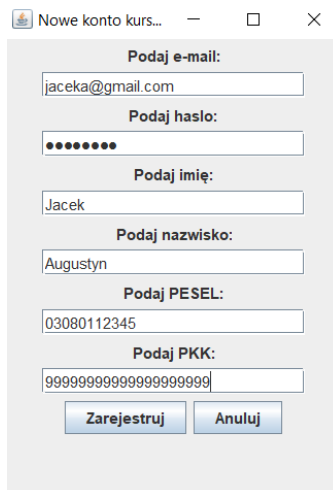
```
@Override
public void actionPerformed(ActionEvent e) {
    Object source = e.getSource();
    if (source == cancel) {
        Window win = SwingUtilities.getWindowAncestor((Window) this);
        ((Window) win).dispose();
    }
    if (source == register) {
        boolean mozna = true;
        String[] napisy = {login.getText(), haslo.getText(), imie.getText(), nazwisko.getText(), pesel.getText(), pkk.getText()};
        for (int i = 0; i < napisy.length; i++)
            if (napisy[i].length() == 0 || napisy[i] == null || napisy[i] == "") {
                mozna = false;
                niezarejestrowano();
            }
        if (mozna == true)
            zarejestrowano();
    }
}
```

Jeżeli chcemy zarejestrować nowego kursanta, program szczytuje teksty z kolejnych pól, i sprawdza czy wszystkie zostały wypełnione. Jeżeli tak, tworzymy nowego kursanta:

```
void zarejestrowano() {
    Window win = SwingUtilities.getWindowAncestor((Window) this);
    ((Window) win).dispose();
    KursanciControler kc = new KursanciControler();
    boolean flaga = kc.add(imie.getText(), nazwisko.getText(), login.getText(), haslo.getText(), pkk.getText(), pesel.getText());
    if (flaga == true) {
        JOptionPane.showMessageDialog(parentComponent: this, message: "Konto zostało utworzone!");
    } else
        JOptionPane.showMessageDialog(parentComponent: this, message: "Wystapil blad. Sprobuj ponownie");
}
```

Następnie dodajemy go do bazy poprzez funkcję *add*:

```
public boolean add(String imie, String nazwisko, String email, String haslo, String pkk, String pesel) {  
  
    entityManager.getTransaction().begin();  
    KursanciEntity kursant = new KursanciEntity();  
    kursant.setImie(imie);  
    kursant.setNazwisko(nazwisko);  
    kursant.setPesel(pesel);  
    kursant.setEmail(email);  
    kursant.setPkk(pkk);  
    kursant.setHaslo(haslo);  
    kursant.setDataRejestracji(new Date(System.currentTimeMillis()));  
    entityManager.persist(kursant);  
    entityManager.getTransaction().commit();  
    return true;  
}
```



Rysunek 17. Przykładowe dane do rejestracji nowego kursanta

```
INSERT INTO KURSANSI  
VALUES (  
    '122',  
    '19/06/10',  
    'jaceka@gmail.com',  
    'jacek123',  
    'Jacek',  
    'Augustyn',  
    '03080112345',  
    '999999999999999999'  
);
```

Rysunek 18. Zapytanie SQL odpowiadające funkcji *add*

Po wykonaniu tej funkcji, tabela **KURSANTCI** uzupełnia się o nowego kursanta:

| | KURSANT_ID | DATA_REJESTRACJI | EMAIL | HASLO | IMIE | NAZWISKO | PESEL | PKK |
|---|------------|------------------|--------------------|------------|---------|-------------|-------------|----------------------|
| 1 | 122 | 19/06/10 | jaceka@gmail.com | jacek123 | Jacek | Augustyn | 03080112345 | 99999999999999999999 |
| 2 | 4 | 19/05/31 | marekg@gmail.com | marek123 | Marek | Piórnik | 98010312345 | 44444444444444444444 |
| 3 | 1 | 19/06/08 | emiliaa@gmail.com | emilia123 | Emilia | Augustyn | 99010512345 | 11111111111111111111 |
| 4 | 2 | 19/05/28 | mateuszs@gmail.com | mateusz123 | Mateusz | Sliwka | 98070112345 | 22222222222222222222 |
| 5 | 5 | 19/04/29 | monikas@gmail.com | monika123 | Monika | Sobierajska | 97060512345 | 55555555555555555555 |
| 6 | 3 | 19/06/01 | joannak@gmail.com | joanna123 | Joanna | Komorniczak | 98040512345 | 33333333333333333333 |

2. Dodawanie rezerwacji przez kursanta

Rysunek 19. Panel obsługi kursanta

Nasz użytkownik nie ma jeszcze żadnej rezerwacji. Aby dodać rezerwację klikamy przycisk *Dodaj rezerwację*

Rysunek 20. Dodawanie rezerwacji

Kiedy uzupełnimy pola odpowiadającymi nam danymi i klikniemy przycisk *Dodaj*, wywołuje się funkcja która sprawdza, czy w danym czasie instruktor ma wolny termin. Wykonuje ona zapytanie do bazy danych:

```
SELECT * FROM REZERWACJE
WHERE INSTRUKTOR_ID=(SELECT INSTRUKTOR_ID FROM INSTRUKTORZY
                      WHERE IMIE='Tomasz' AND NAZWISKO='Cudny');
```

Rysunek 21. Pobranie wszystkich rezerwacji na danego instruktora

Następnie funkcja w pętli dodaje wszystkie daty rezerwacji na instruktora do dat, które są zajęte. W tym celu funkcja musi zapytać bazę o daty wszystkich rezerwacji na instruktora:

```
SELECT DATA_REZERWACJI FROM REZERWACJE
WHERE INSTRUKTOR_ID=(SELECT INSTRUKTOR_ID FROM INSTRUKTORZY
                      WHERE IMIE='Tomasz' AND NAZWISKO='Cudny');
```

Rysunek 22. Pobranie dat rezerwacji na danego instruktora

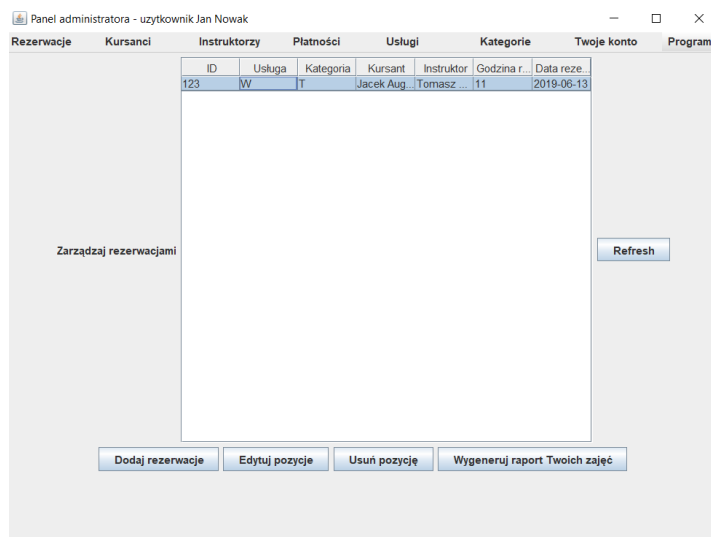
Analogicznie funkcja postępuje z godzinami rezerwacji. Następnie sprawdza, czy podana przez nas data i godzina znajdują się na listach. Jeśli nie, rezerwacja zostaje dodana do bazy danych.

```
INSERT INTO REZERWACJE
VALUES (
'123',
(SELECT CURRENT_DATE FROM DUAL),
'19/06/13',
'11',
(SELECT INSTRUKTOR_ID FROM INSTRUKTORZY
 WHERE NAME='Tomasz' AND NAZWISKO='Cudny'),
(SELECT KATEGORIA_ID FROM KATEGORIE
 WHERE SYMBOL='T'),
'122',
(SELECT USLUGA_ID FROM USLUGI
 WHERE NAZWA='W')
);
```

Rysunek 23. Dodawanie rezerwacji do bazy danych

3. Edycja rezerwacji

Logując się jako instruktor, który ma uprawnienia admina, możemy edytować wszystkie rezerwacje jakie są w systemie. W tym celu zaznaczamy pożądaną rezerwację i klikamy przycisk *Edytuj pozycję*:



Rysunek 24. Panel obsługi instruktora-admina

The screenshot shows a form titled 'Edycja rezerwacji'. It contains the following fields and controls:

- 'Wybierz kursanta:' with a dropdown menu showing 'Jacek Augustyn'.
- 'Wybierz usługę:' with a dropdown menu showing 'X'.
- 'Wybierz kategorię:' with a dropdown menu.
- 'Wybierz instruktora:' with a dropdown menu showing 'Tomasz Cudny'.
- 'Wybierz datę Twojej nowej rezerwacji:' with three dropdown menus for day (05), month (07), and year (2019).
- 'Wybierz godzinę rozpoczęcia:' with a dropdown menu showing '8'.
- Two buttons at the bottom: 'Aktualizuj' and 'Anuluj'.

Rysunek 25. Okienko edycji rezerwacji

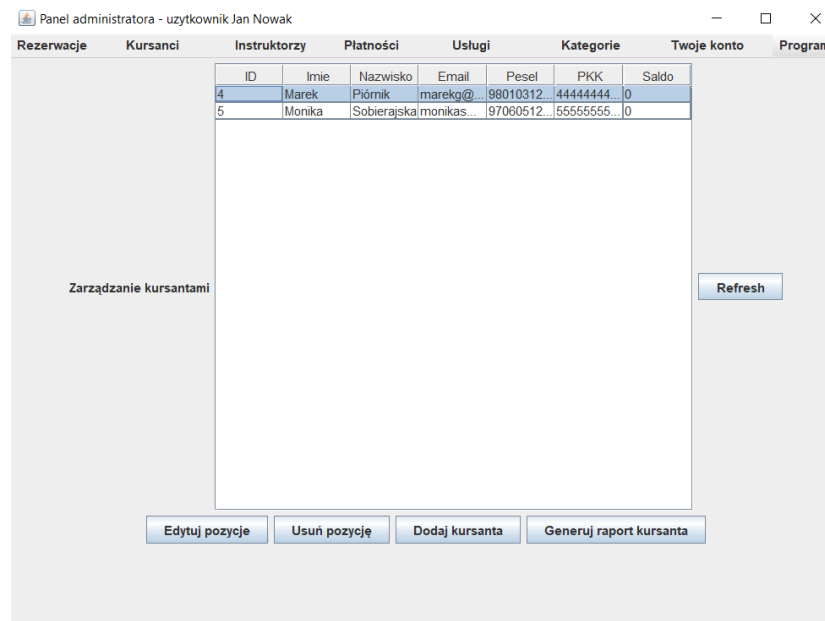
Kiedy wybierzemy nowe dane, funkcja aktualizująca *update* rezerwację wykonuje zapytanie do bazy:

```
UPDATE REZERWACJE
SET INSTRUKTOR_ID = (SELECT INSTRUKTOR_ID FROM INSTRUKTORZY
                     WHERE IMIE='Tomasz' AND NAZWISKO='Cudny'),
KURSANT_ID = (SELECT KURSANT_ID FROM KURSANTI
              WHERE IMIE='Jacek' AND NAZWISKO='Augustyn'),
USLUGA_ID = (SELECT USLUGA_ID FROM USLUGI
              WHERE NAZWA='X'),
DATA_DODANIA = (SELECT CURRENT_DATE FROM DUAL),
GODZ_ROZPOCZECIA = '8',
KATEGORIA_ID = (SELECT KATEGORIA_ID FROM KATEGORIE
                 WHERE SYMBOL='T'),
DATA_REZERWACJI = '19/07/05'
WHERE REZERWACJA_ID='123';
```

Następnie odświeża listę rezerwacji, pobierając z bazy zaktualizowaną listę rezerwacji dla danego instruktora:

```
SELECT * FROM REZERWACJE
WHERE INSTRUKTOR_ID = '106';
```

4. Usuwanie kursanta po ID



Rysunek 26. Panel obsługi instruktora-admina

Aby usunąć kursanta po ID, zaznaczamy odpowiednią pozycję i klikamy przycisk *Usuń pozycję*. Pobierane jest wtedy ID kursanta i wywołuje się funkcja *deleteByID*, która usuwa kursanta kaskadowo razem ze wszystkimi jego rezerwacjami i płatnościami.

```

public void deleteByID(long id) {
    entityManager.getTransaction().begin();
    KursanciEntity ke = entityManager.find(KursanciEntity.class, id);
    entityManager.remove(ke);
    entityManager.getTransaction().commit();
}

```

Rysunek 27. Funkcja usuwająca kursanta

Funkcja *deleteByID* wykonuje poniższe zapytanie do bazy danych:

```

DELETE FROM PLATNOSCI
WHERE KURSANT_ID='4';
DELETE FROM REZERWACJE
WHERE KURSANT_ID='4';
DELETE FROM KURSANCY
WHERE KURSANT_ID='4';

```

Lista kursantów odświeża się poprzez funkcję *refreshList*, która wykonuje zapytanie:

```

SELECT KURSANT_ID, IMIE, NAZWISKO, EMAIL, PESEL, PKK
FROM KURSANCY;

```

W wyniku tych działań lista kursantów wygląda teraz tak:

Panel administratora - użytkownik Jan Nowak

RezerwacjeKursanciInstruktorzyPłatnościUsługiKategorieTwoje kontoProgram

Zarządzanie kursantami

| ID | Imie | Nazwisko | Email | Pesel | PKK | Saldo |
|----|--------|-------------|------------|-------------|-------------|-------|
| 5 | Monika | Sobierajska | monikas... | 97060512... | 55555555... | 0 |

Refresh

Edytuj pozycje

Usuń pozycję

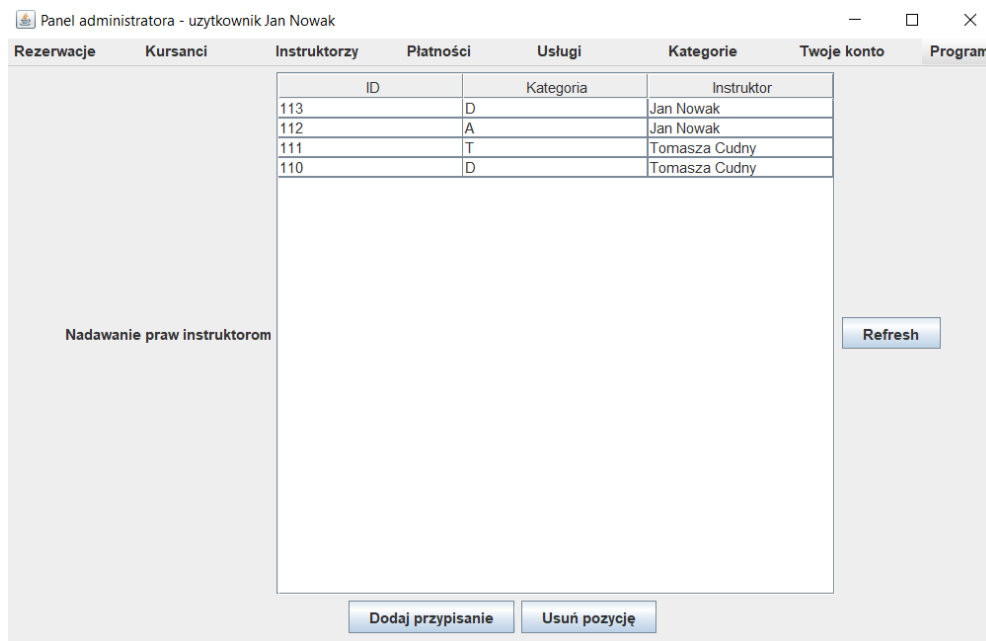
Dodaj kursanta

Generuj raport kursanta

Rysunek 28. Lista kursantów po usunięciu kursanta o id = 4

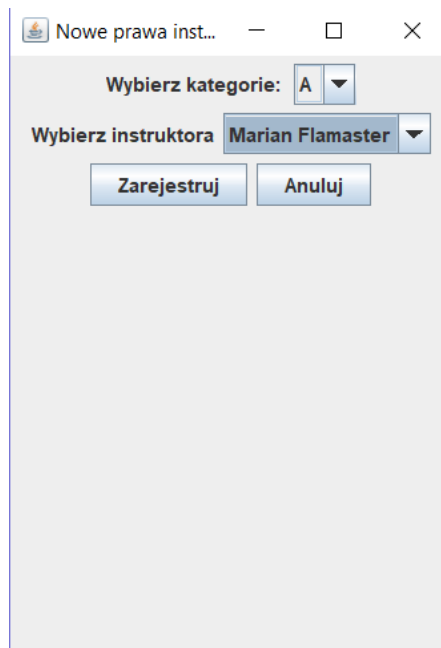
5. Przypisywanie kategorii instruktorom

Logując się jako instruktor przechodzimy do zakładki *Kategorie*. Wyświetla nam się lista instruktorów z przyporządkowanymi im kategoriami.



Rysunek 29. Panel obsługi instruktora

Możemy dodać nowe przypisanie klikając przycisk *Dodaj przypisanie*.



Rysunek 30. Przypisywanie kategorii do instruktora

Kiedy klikniemy przycisk *Zarejestruj*, funkcja *add* dodaje do tabeli **KATEGORIE_INSTRUKTOROW** podane przez nas „połączenie”. **ID_WPISU** generuje się automatycznie. Dodanie nowego „połączenia” wykonuje się poprzez poniższe zapytanie:

```
INSERT INTO KATEGORIE_INSTRUKTOROW
VALUES(
'141',
(SELECT INSTRUKTOR_ID FROM INSTRUKTORZY
WHERE IMIE='Marian' AND NAZWISKO='Flamaster'),
(SELECT KATEGORIA_ID FROM KATEGORIE
WHERE SYMBOL='A')
);
```

Po dodaniu nowego połączenia lista odświeża się poprzez funkcję *refreshList*, która wykonuje zapytanie:

```
SELECT ID_WPISU, SYMBOL, IMIE, NAZWISKO
FROM KATEGORIE_INSTRUKTOROW
JOIN KATEGORIE ON KATEGORIE_INSTRUKTOROW.KATEGORIA_ID=KATEGORIE.KATEGORIA_ID
JOIN INSTRUKTORZY ON INSTRUKTORZY.INSTRUKTOR_ID=KATEGORIE_INSTRUKTOROW.INSTRUKTOR_ID;
```

Panel administratora - użytkownik Jan Nowak

RezerwacjeKursanciInstruktorzyPłatnościUsługiKategorieTwoje kontoProgram

Nadawanie praw instruktorom

| ID | Kategoria | Instruktor |
|-----|-----------|------------------|
| 113 | D | Jan Nowak |
| 112 | A | Jan Nowak |
| 111 | T | Tomasza Cudny |
| 110 | D | Tomasza Cudny |
| 141 | A | Marian Flamaster |

Refresh

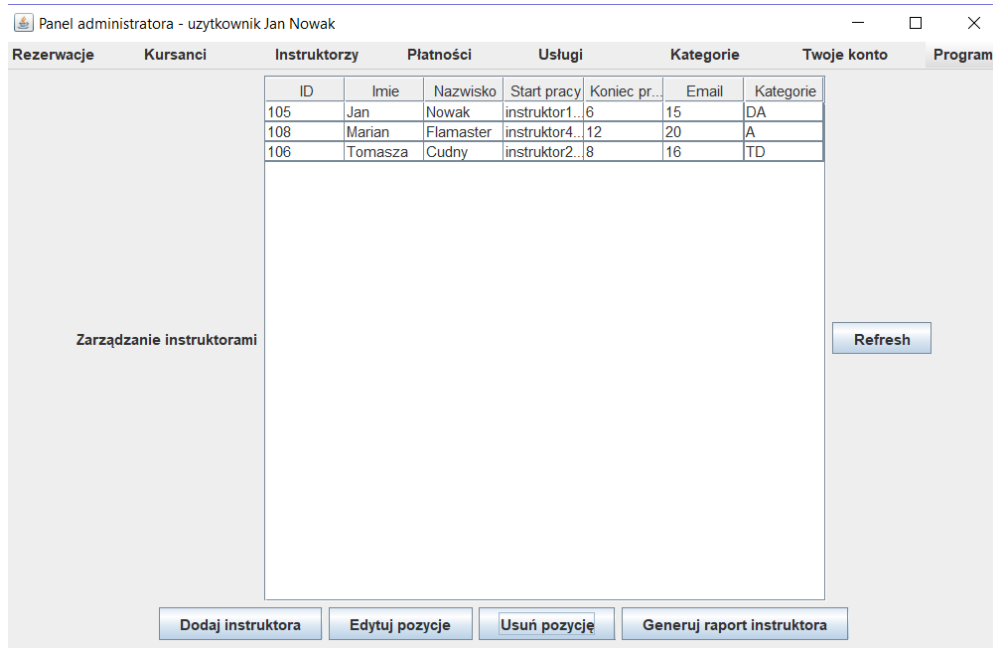
Dodaj przypisanie

Usuń pozycję

Rysunek 31. Kategorie instruktorów po dodaniu nowego połączenia

6. Usuwanie instruktora

Aby usunąć instruktora, logujemy się jako instruktor z prawami admina i przechodzimy do zakładki *Instruktorzy*:



Rysunek 32. Panel obsługi instruktora

Wybieramy instruktora, którego chcemy usunąć i klikamy przycisk *Usuń pozycję*. Wywołuje się wtedy funkcja *deleteByID*, która usuwa instruktora kaskadowo razem ze wszystkimi jego rezerwacjami i kategoriami (podobnie jak przy usuwaniu kursanta).

```
public void deleteByID(long id) {  
    entityManager.getTransaction().begin();  
    InstruktorzyEntity ke = entityManager.find(InstruktorzyEntity.class, id);  
    entityManager.remove(ke);  
    entityManager.getTransaction().commit();  
}
```

Rysunek 33. Funkcja usuwająca instruktora

Usuwanie instruktora dokonuje się poprzez wywołanie zapytań:

```
DELETE FROM REZERWACJE  
WHERE INSTRUKTOR_ID='105';  
DELETE FROM KATEGORIE_INSTRUKTOROW  
WHERE INSTRUKTOR_ID='105';  
DELETE FROM INSTRUKTORZY  
WHERE INSTRUKTOR_ID='105';
```

Następnie lista instruktorów odświeża się poprzez funkcję *refreshList*, która wykonuje zapytanie:

```
SELECT INSTRUKTOR_ID, IMIE, NAZWISKO, EMAIL, GODZ_ROZPOCZECIA, GODZ_ZAKONCZENIA, SYMBOL  
FROM INSTRUKTORZY  
JOIN KATEGORIE_INSTRUKTOROW ON INSTRUKTORZY.INSTRUKTOR_ID=KATEGORIE_INSTRUKTOROW.INSTRUKTOR_ID  
JOIN KATEGORIE ON KATEGORIE_INSTRUKTOROW.KATEGORIA_ID=KATEGORIE.KATEGORIA_ID;
```

Po usunięciu instruktora możemy łatwo zauważyć, że nie możemy się na niego już zalogować:

Rysunek 34. Błąd logowania - brak instruktora w bazie

2.6 Podsumowanie

a) Założenia

Podczas pracy nad produkcją interfejsu zdecydowaliśmy się na małą zmianę w architekturze zależności między użytkownikiem a płatnościami. Zamiast przypisywać płatność do każdej jego rezerwacji z osobna zdecydowaliśmy się na powiązanie płatności bezpośrednio z kontem użytkownika tj. płatność przypisywana jest do ID użytkownika (klucz obcy). Pozwala to na sprawniejsze operowanie interfejsem i szybsze przypisywanie wpłat (nie trzeba szukać konkretnej rezerwacji, wystarczy wskazać użytkownika, który dokonuje wpłaty). Obciążenia natomiast naliczane są automatycznie dzięki powiązaniu rezerwacji z tabelą usług (każda z usług ma swoją cenę). Po przeliczeniu rezerwacji oraz wpłat uzyskujemy saldo kursanta.

Reszta założeń została spełniona wg. opisu problemu w podpunkcie *Przedstawienie problemu* (Ctrl+kliknięcie śledzi łączy).

b) Wnioski

Baza danych Oracle w wersji 11 nie wspiera auto inkrementacji ID przy przypisywaniu ich do nowych rekordów. Aby rozwiązać ten problem, generowano w zapytaniach nowe ID poprzez funkcję COUNT(), albo opierano się na mechanizmie Hibernate. Mechanizm ten generuje listę zbiorczych ID dla wszystkich encji, przez co w całej bazie obowiązuje jedna, ogólna lista ID, a nie osobne listy dla każdej encji.

Podczas uzupełniania bazy danych należy uważać na zgodność typów i możliwość kolizji pojedynczych rekordów, np. nakładania się godzin rezerwacji. Rozwiązano to generując szereg zapytań (opisane w punkcie Selekcja/prezentacja – SELECT). Na podstawie tych zapytań napisano funkcję, która podczas dodawania nowych rezerwacji przy pomocy interfejsu sprawdza ich poprawność i dopiero wtedy wykonuje transakcję z bazą danych.

Chęć stworzenia interfejsu zgodnego z zasadą 'userfriendly' zmusiła nas do sporządzenia kilku dodatkowych zapytań, które np. zamiast „suchego” ID instruktora zwracało jego imię i nazwisko, czego przykładem jest formularz dodawania rezerwacji w panelu kursanta/administradora.

3. Literatura

[1] <https://www.samouczekprogramisty.pl/wstep-do-relacyjnych-baz-danych/#czym-jest-baza-danych>

[2] http://www.metal.agh.edu.pl/~regulski/bd-podyp/00-wyklady/05_model_relacyjny.pdf

[3] <https://www.sqlpedia.pl/relacyjne-bazy-danych-pojecia-podstawowe/>

[4] <https://www.sqlpedia.pl/projektowanie-i-normalizacja-bazy-danych/>

[5] <https://zeszyt.jedlikowski.com/2014-10-27/sbd/postulaty-codda-dotyczace-relacyjnych-baz-danych-rodzaje-kluczy-zasady-dobierania-kluczy-powiazania-pomiedzy-relacjami/>

[6] <https://www.w3schools.com/sql/>