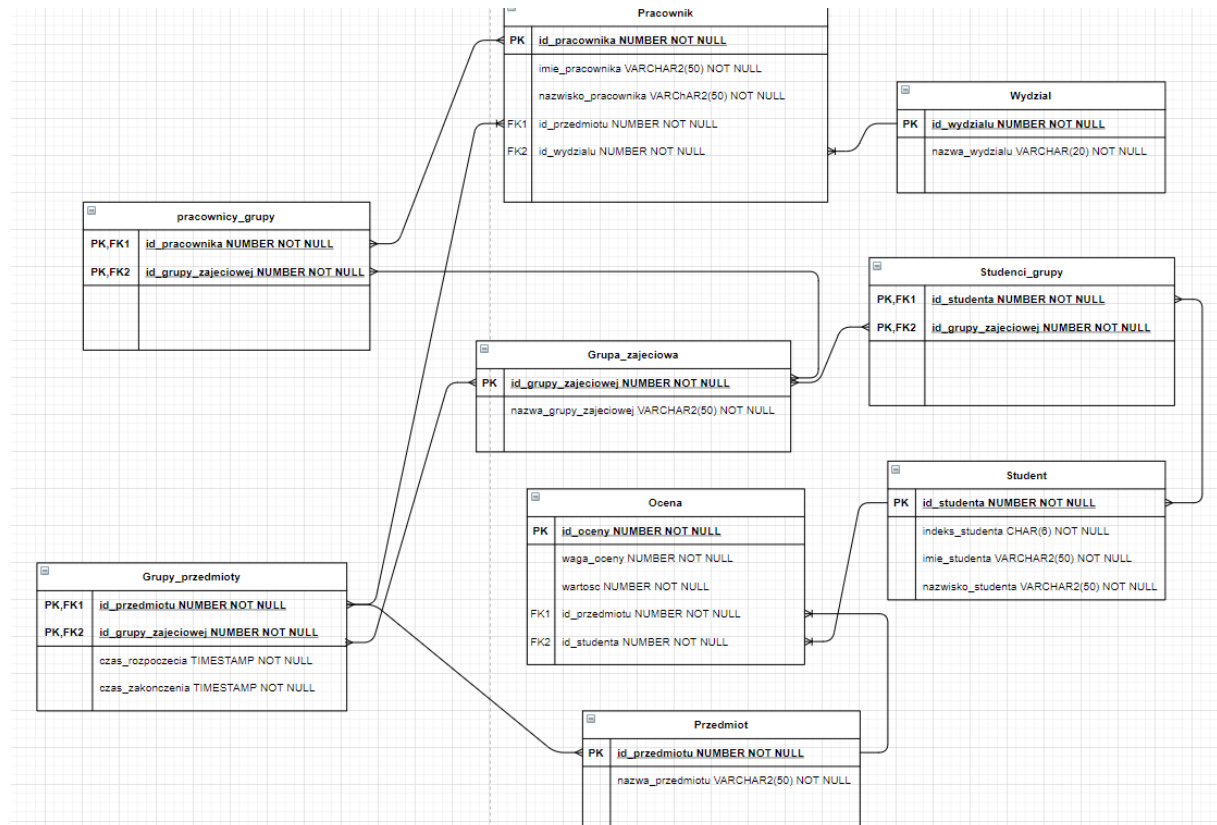


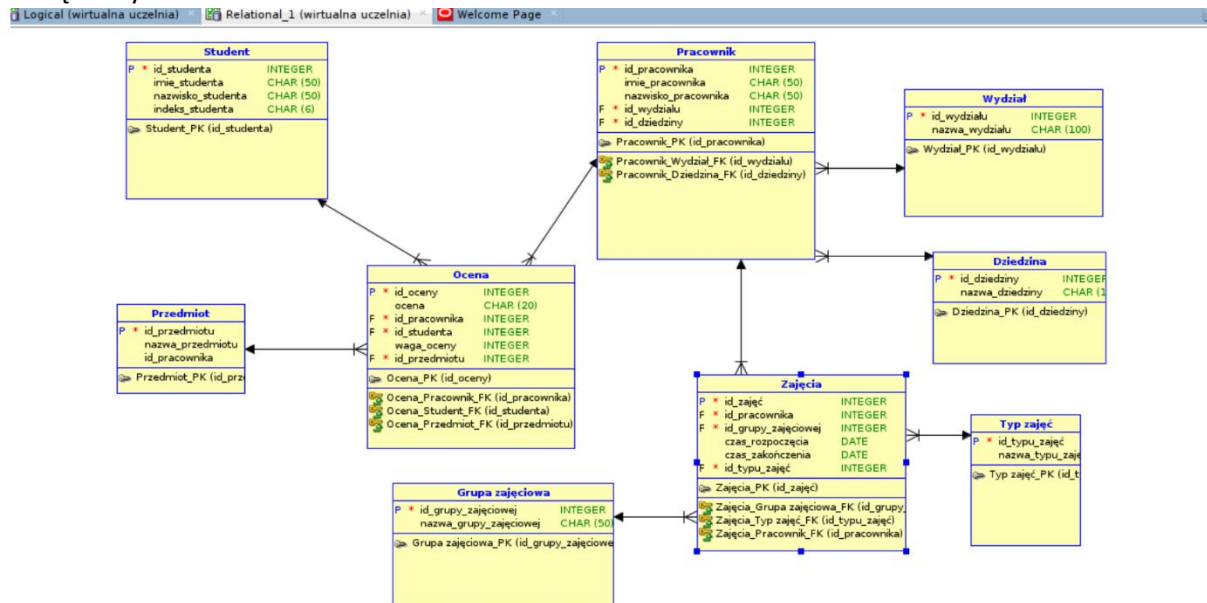
Łukasz Dyrłaga
Informatyka Techniczna, semestr 4,
rok akademicki 2020 - 2021
grupa projektowa nr. 1
indeks: 400222

Schemat z draw.io

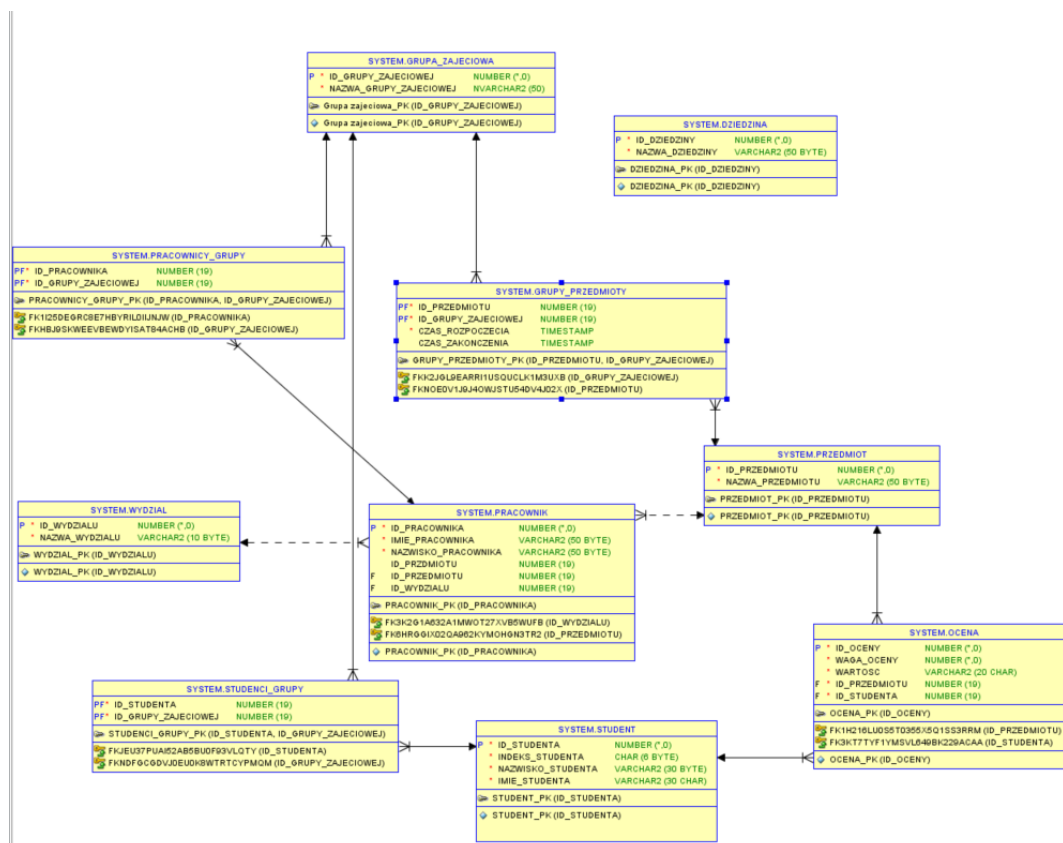


W projekcie każdy student należy do 1 lub więcej grup (laboratoryjna, audytoryjna, wykładowa) dlatego występuje relacja wiele do wielu, w każdej grupie powinna być +/- ta sama liczba studentów, dlatego napisana została procedura podziału studentów do grup. Każdy student może też mieć wiele ocen z wielu przedmiotów, dlatego jest relacja 1 do wielu z ocenami. Każda ocena ma wagę 1 (ćwiczenia, laboratoria) lub 3 (egzamin), ocena posiada również rodzaj przedmiotu, z którego została wystawiona, każdy przedmiot może mieć wiele ocen, występuje relacja 1 do wielu. Przedmioty mogą mieć wiele grup zajęciowych, występuje relacja wiele do wielu. Pracownicy mogą należeć do 1 wydziału więc jeden wydział może mieć wielu pracowników, relacja 1 do wielu. Pracownik może mieć wiele grup zajęciowych, występuje relacja 1 do wielu.

Początkowy schemat:



Aktualny:



Większą zmianą jest usunięcie tabeli **TypZajec** nie była ona użyteczna, dodanie relacji wiele do wielu między pracownikami a grupami, studentami a grupami, przedmiotami a grupami. Zmiana typów date na timestamp aby wskazywały też godziny. Dodanie sekwencji, inkrementujących wartości kluczy podstawowych w każdej tabeli. Tablica dziedzina może być użyta w przyszłości do sortowania pracowników po specjalizacji, której mogą uczyć.

Procedura dodająca do tabeli **grupy_przedmioty** będącej tabelą pomocniczą przechowującą informacje o relacjach **wiele do wielu** między grupami zajęciowymi a przedmiotami. Wiele przedmiotów może mieć wiele grup zajęciowych. Do tabeli dodawana jest informacja o **id_grupy_zajeciowej**, **id_przedmiotu** oraz **czas_roz poczenia** pierwszych zajęć, wszystkie zajęcia są cyklicznie **CO TYDZIEŃ**.

```
create or replace PROCEDURE
PROCEDURE_ADD_GRUPA_PRZEDMIOT(id_grupy_zajeciowej IN NUMBER,
id_przedmiotu IN NUMBER, czas_roz poczenia IN Date) AS

BEGIN
    INSERT INTO grupy_przedmioty (id_grupy_zajeciowej, id_przedmiotu,
    czas_roz poczenia)
        VALUES (id_grupy_zajeciowej, id_przedmiotu,
    czas_roz poczenia);
END PROCEDURE_ADD_GRUPA_PRZEDMIOT;
```

Trigger działający we współpracy z poprzednio działającą procedurą, sprawdza czy termin zajęć nie wskazuje na sobotę albo niedzielę, w takim przypadku wyrzucony zostaje błąd numer **-20000**. W przeciwnym wypadku, dodana zostaje dodatkowa wartość zakończenia zajęć, zajęcia trwają zawsze **1.5h**, do sprawdzanej w triggerze wartości **czas_roz poczenia**, dodane zostaje **1.5h**.

```
create or replace TRIGGER TRIGGER2

BEFORE INSERT OR UPDATE OF CZAS_ROZPOCZECIA ON grupy_przedmioty
FOR EACH ROW
BEGIN
    DBMS_OUTPUT.put_line('Is working');
    IF to_char(:New.czas_roz poczenia, 'D') >= 6 THEN
        raise_application_error( -20000, 'error during saving
grupa_zajeciowa');
    ELSE
        :New.czas_zakonczenia := :New.czas_roz poczenia + interval '30'
minute + interval '1' hour;
        /* INSERT INTO LOGS (column1, column2) VALUES
(to_char(:New.czas_roz poczenia, 'D'), :New.czas_roz poczenia - interval
'30' minute);*/
    END IF;
END;
```

Procedura dodająca studenta do grupy zajęciowej. Ponieważ występuje relacja **wiele do wielu**, oba id zostają dodane do pomocniczej tabeli **studenci_grupy**.

```
create or replace PROCEDURE PROCEDURE_ADD_GRPUPA_STUDENT(id_studenta IN
NUMBER, id_grupy_zajeciowej IN NUMBER) AS

    groups_num NUMBER;
BEGIN
    dbms_output.put_line(id_grupy_zajeciowej);
    INSERT INTO STUDENCI_GRPUPY (ID_STUDENTA, ID_GRPUPY_ZAJECIOWEJ)
        VALUES
            (id_studenta, id_grupy_zajeciowej);
END PROCEDURE_ADD_GRPUPA_STUDENT;
```

CTE, FETCH

Funkcja przy użyciu **CTE** oraz **FETCH** (`rownum < liczba`) zwraca pierwsze **10%** wierszy zawierających **id_studenta** oraz jego średnią ocen obliczoną według wzoru:

$$\frac{\sum waga * ocena}{\sum waga}$$

Funkcja pobiera ilość studentów do zmiennej **students_num** i za pomocą klauzuli **WHERE** `rownum < students_num / 10`, zwraca tylko pierwsze 10% wierszy posortowanych malejąco po średniej ocen, czyli tylko najlepszych studentów którzy dostaną stypendium.

W wersji **11g** niestety nie da się skorzystać z klauzul **FETCH FIRST, OFFSET**.

Odpowiednikiem :

```
SELECT * FROM avg_CTE WHERE rownum < students_num / 10;
```

Powinno być zapytanie:

```
SELECT * FROM avg_CTE FETCH FIRST students_num / 10 ROWS ONLY;
```

```
create or replace FUNCTION FUNCTION_AVG_OF_OCENY_ALL RETURN
SYS_REFCURSOR AS
  c1 SYS_REFCURSOR;
  students_num NUMBER;
BEGIN
  SELECT count(*) INTO students_num FROM student;

  OPEN c1 FOR
  WITH oceny_CTE AS
  (
    SELECT o.id_studenta, o.wartosc * o.waga_oceny AS suma,
    o.waga_oceny AS waga FROM ocena o
  ),
  avg_CTE AS
  (
    SELECT id_studenta, SUM(suma) / SUM(waga) AS srednia FROM
    oceny_CTE GROUP BY id_studenta ORDER BY (SUM(suma) / SUM(waga)) DESC
  )
  SELECT * FROM avg_CTE WHERE rownum < students_num / 10;

  RETURN(c1);
END FUNCTION_AVG_OF_OCENY_ALL;
```

Przykładowy wynik:

ID_STUDE...	SREDNIA
25	4,75
10	4,5
1	4,4285714...

Funkcja zwracają wynikowy zbiór wierszy, które przedstawiają **id_studenta** oraz **średnią** jego ocen z podanego przez nas **id_przedmiotu**.

```
create or replace FUNCTION FUNCTION_AVG_OF_OCENY(id_przedmiotu NUMBER)
RETURN SYS_REFCURSOR AS

    c1 SYS_REFCURSOR;
BEGIN
    OPEN c1 FOR
    WITH oceny_CTE AS
    (
        SELECT o.id_studenta, o.wartosc * o.waga_oceny AS suma,
        o.waga_oceny AS waga FROM ocena o WHERE o.id_przedmiotu = id_przedmiotu
    )
        SELECT id_studenta, SUM(suma) / SUM(waga) AS srednia FROM oceny_CTE
        GROUP BY id_studenta;

    RETURN(c1);
END FUNCTION_AVG_OF_OCENY;
```

Przykładowy wynik:

ID_STUDENTA	SREDNIA
1	4,42857142857142...
22	3
25	4,75
6	3,4
13	2,4
11	2,2
2	4
14	2,6
20	3,75
21	3,33333333333333...
4	3,25
5	3,75
24	3
23	3
17	4
8	2
3	3,08333333333333...
18	2,57142857142857...
7	2,85714285714285...
12	3,14285714285714...
19	3
16	3,33333333333333...
10	4,5
15	3
9	2,75

Funkcja zwracają wynikowy zbiór wierszy, które przedstawiają **id_studenta** oraz **średnią** jego ocen z podanego przez nas **id_przedmiotu**, dodatkowo użyta została klauzula **MINUS** zwracają występujące **tylko i wyłącznie** w pierwszym selekcje wartości, w tym przypadku zwrócone zostaną id'ki studentów, którzy nie zdali.

```
create or replace FUNCTION FUNCTION_WEAK_OCENY(id_przedmiotu NUMBER)
RETURN SYS_REFCURSOR AS
    c1 SYS_REFCURSOR;
BEGIN
    OPEN c1 FOR
    WITH oceny_CTE AS
    (
        SELECT o.id_studenta, o.wartosc * o.waga_oceny AS suma,
        o.waga_oceny AS waga FROM ocena o WHERE o.id_przedmiotu = id_przedmiotu
    )
    SELECT id_studenta, SUM(suma) / SUM(waga) AS srednia FROM oceny_CTE
    GROUP BY id_studenta
    MINUS
    SELECT id_studenta, SUM(suma) / SUM(waga) AS srednia FROM oceny_CTE
    GROUP BY id_studenta HAVING SUM(suma) / SUM(waga) > 3;

    RETURN(c1);
END FUNCTION_WEAK_OCENY;
```

Przykładowy wynik:

ID_STUDE...	SREDNIA
7	2,8571428...
8	2
9	2,75
11	2,2
13	2,4
14	2,6
18	2,5714285...

Procedura zwracająca do kursora **s_cursor** wynikowy zbiór wierszy zawierający wszystkie oceny dla studenta o danym **id**, do złączenia obu tabel użyto **LEFT JOIN** za pomocą **id_studenta**.

```
create or replace PROCEDURE PROCEDURE_GET_OCENY (id IN NUMBER, s_cursor
OUT SYS_REFCURSOR)
IS
BEGIN
    IF id != 0 THEN
        OPEN s_cursor FOR
        SELECT nazwa_przedmiotu, wartosc
        FROM view_get_oceny
        LEFT JOIN student s
        ON s.id_studenta = id;

    END IF;

END PROCEDURE_GET_OCENY;
```

Widok **view_get_oceny**:

```
SELECT
    p.nazwa_przedmiotu, o.wartosc
FROM przedmiot p
LEFT JOIN ocena o
    ON p.id_przedmiotu = o.id_przedmiotu
```

Przykładowy wynik

NAZWA_P...	WARTOSC
Algebra	4
Algebra	2
Algebra	2
Algebra	3
Algebra	2
Algebra	3
Algebra	4
Podstawy ...	3
Podstawy ...	2
Podstawy ...	2
Podstawy ...	2
Podstawy ...	2
Podstawy ...	4
Podstawy ...	5
Podstawy ...	5
Podstawy ...	4
Podstawy ...	4
Podstawy ...	5
Bazy danych	5
Bazy danych	5
Bazy danych	5
Bazy danych	4
Bazy danych	4
Bazy danych	2
Bazy danych	5
Bazy danych	2
Bazy danych	3
Bazy danych	2
Bazy danych	3
Bazy danych	3
Bazy danych	3
Algebra	5
Podstawy ...	4
Bazy danych	3
Programo...	3
Programo...	3
Programo...	3

Przykładowy widok z użyciem klauzuli **INTERSECT**, zwracającej jedynie te wiersze, które są wspólne dla obu selectów.

```
SELECT gz.id_grupy_zajeciowej FROM grupa_zajeciowa gz  
  
INTERSECT  
SELECT sg.id_grupy_zajeciowej FROM studenci_grupy sg  
WHERE sg.id_studenta = 4
```

Przykładowy wynik:

ID_GRUPY_ZAJECIOWEJ	
1	1
2	5
3	7

```

create or replace PROCEDURE PROCEDURE_DIVIDE_STUDENTS (group_type IN
varchar2) AS

    students_num NUMBER;
    groups_num NUMBER;
    in_group_num INTEGER;
    mod_value NUMBER;
    counter INTEGER default 0;
    i INTEGER default 0;
    id_grupy NUMBER default 0;
    id_studenta NUMBER;
BEGIN

    SELECT COUNT(s.id_studenta) INTO students_num FROM student s;

    SELECT COUNT(gz.id_grupy_zajeciowej) INTO groups_num FROM
grupa_zajeciowa gz
    WHERE LOWER(gz.nazwa_grupy_zajeciowej) LIKE LOWER(CONCAT(group_type,
'%'));

    in_group_num := students_num / groups_num;
    mod_value := students_num mod groups_num;

    LOOP
        /*INSERT INTO GRUPA_ZAJECIOWA*/

        IF (counter mod in_group_num = 0) THEN
            i := i + 1;

            dbms_output.put_line(counter);
            dbms_output.put_line(i);
            SELECT gz.id_grupy_zajeciowej INTO id_grupy
            FROM grupa_zajeciowa gz
            WHERE LOWER(gz.nazwa_grupy_zajeciowej)
            LIKE LOWER(CONCAT(CONCAT(group_type, to_char(i)), '%'));

        END IF;
        counter := counter + 1;

        EXIT WHEN
            i > groups_num;

        EXIT WHEN counter = in_group_num * groups_num;
        procedure_add_grupa_student(counter, id_grupy);

    END LOOP;

END PROCEDURE_DIVIDE_STUDENTS;

```

Procedura rozdzielająca studentów do poszczególnych grup, zwraca ilość studentów i grup_zajęciowych danego typu (lab, audyt, wykład), oblicza ile w danej grupie powinno być studentów. W pętli zwiększane sprawdzane są id danych grup, i przy założeniu że id studentów są w przedziale 1...n, wstawiani są oni do poszczególnych grup. Procedura podczas testów działała poprawnie, dodatkowo można wspomóc jej działanie o rozdzielanie takiej ilości osób, przy której nie uda się mieć *k* identycznych ilościowo grup, wtedy można by tak zmienić procedurę aby dodała po 1 dodatkowej osobie do pierwszych *j* grup.

Grupowanie z użyciem klauzuli **ROLLUP()**, dzięki rollup wykonane zostają:

GROUP BY(p.nazwa_przedmiotu, o.wartosc)

GROUP BY(p.nazwa_przedmiotu)

GROUP BY()

```
SELECT p.nazwa_przedmiotu, SUM(o.wartosc * o.waga_oceny) /
SUM(o.waga_oceny) AS srednia , COUNT(*) liczba
FROM przedmiot p
LEFT JOIN ocena o
ON p.id_przedmiotu = o.id_przedmiotu
GROUP BY ROLLUP(p.nazwa_przedmiotu, o.wartosc);
```

Dla każdego przedmiotu policzono ilość odpowiednich ocen, oraz średnie. Na końcu policzono średnią dla wszystkich przedmiotów i sume ocen.

Przykładowy wynik: (przedmiot, ocena, liczność tej oceny)

16 Grafika komputerowa	4	15
17 Analiza matematyczna	2	3
18 Analiza matematyczna	3	6
19 Analiza matematyczna	4	4
20 Analiza matematyczna	5	2
21 Analiza matematyczna	3,36842105263157894736842105263157894737	15
22 Podstawy informatyki	2	4
23 Podstawy informatyki	3	1
24 Podstawy informatyki	4	4
25 Podstawy informatyki	5	4
26 Podstawy informatyki	3,8	13
27 Sztuczna inteligencja	2	2
28 Sztuczna inteligencja	3	3
29 Sztuczna inteligencja	4	6
30 Sztuczna inteligencja	5	2
31 Sztuczna inteligencja	3,4	13
32 Programowanie obiektowe	2	3
33 Programowanie obiektowe	3	6
34 Programowanie obiektowe	4	1
35 Programowanie obiektowe	3	10
36 (null)	3,25	101

Wersja z użyciem klauzuli **CUBE**, dodatkowo następuje pogrupowanie dla danych wartości ocen łącznie z każdego przedmiotu, mamy 26 dwójek, 31 trójek 24 czwórki i 19 piątek. Nulle przy **systemach wbudowanych** mogą być spowodowane brakiem jakichkolwiek ocen z tego przedmiotu.

```
SELECT p.nazwa_przedmiotu, SUM(o.wartosc * o.waga_oceny) /
SUM(o.waga_oceny) AS srednia , COUNT(*) liczba

FROM przedmiot p
LEFT JOIN ocena o
ON p.id_przedmiotu = o.id_przedmiotu
GROUP BY CUBE(p.nazwa_przedmiotu, o.wartosc);
```

NAZWA_PRZEDMIOTU	SREDNIA	LICZBA
1 (null)	(null)	1
2 (null)	3,25	101
3 (null)	2	26
4 (null)	3	31
5 (null)	4	24
6 (null)	5	19
7 Algebra	2,80769230769230769230769230769231	14
8 Algebra	2	7
9 Algebra	3	3
10 Algebra	4	2
11 Algebra	5	2
12 Bazy danych	2,83333333333333333333333333333333	20
13 Bazy danych	2	7
14 Bazy danych	3	7
15 Bazy danych	4	2
16 Bazy danych	5	4
17 Systemy wbudowane	(null)	1
18 Systemy wbudowane	(null)	1
19 Grafika komputerowa	4	15
20 Grafika komputerowa	3	5
21 Grafika komputerowa	4	5
22 Grafika komputerowa	5	5
23 Analiza matematyczna	3,36842105263157894736842105263157894737	15

Tabela oceny (widać brak ocen dla przedmiotu o id: 8 czyli systemów wbudowanych)

ID_OCENY	WAGA_OCENY	WARTOSC	ID_PRZEDMIOTU	ID_STUDENTA
63	63	14	16	16
64	64	13	1	1
65	65	13	15	15
66	66	13	15	15
67	67	13	15	15
68	68	12	11	11
69	69	12	12	12
70	70	12	22	22
71	71	14	22	22
72	72	14	22	22
73	73	14	22	22

Zestawienie ilości ocen do ich wartości za pomocą klauzuli **pivot**.

```
SELECT * from (
    SELECT wartosc
    FROM ocena
)
pivot
(
    count(*) for wartosc in (2, 3, 4, 5)
)
```

Przykładowy wynik:

	2	3	4	5
1	26	31	24	19

Wypisanie liczności kolejnych grup zajęciowych:

```
SELECT * from (
    SELECT gz.nazwa_grupy_zajeciowej
    FROM grupa_zajeciowa gz
    INNER JOIN
    studenci_grupy sg
    ON sg.id_grupy_zajeciowej = gz.id_grupy_zajeciowej
    INNER JOIN
    student s
    ON s.id_studenta = sg.id_studenta
)
pivot
(
    count(*) for nazwa_grupy_zajeciowej in ('lab1IT', 'lab2IT',
    'lab3IT', 'lab4IT', 'audyt1IT', 'audyt2IT', 'wyklad1IT')
)
```

Przykładowy wynik:

	'lab1IT'	'lab2IT'	'lab3IT'	'lab4IT'	'audyt1IT'	'audyt2IT'	'wyklad1IT'
1	6	6	6	6	13	12	25

Spring Boot

CRUD został wykonany za pomocą frameworka **Spring Boot**. Na samym początku należało do pliku **build.gradle** wpisać zależności:

```
dependencies {  
    implementation 'org.springframework.boot:spring-boot-starter-data-jpa'  
    implementation 'org.springframework.boot:spring-boot-starter-web'  
    implementation 'org.springframework.boot:spring-boot-starter-data-jdbc'  
    runtimeOnly 'com.oracle.database.jdbc:ojdbc8'  
    testImplementation 'org.springframework.boot:spring-boot-starter-test'  
}
```

Dzięki temu gradle sam sam zaciągnie odpowiednie zależności, nie będziemy musieli się martwić o samodzielną instalację odpowiednich bibliotek itd.

W pliku konfiguracyjnym **application.properties** umieszczamy informacje o łączeniu się z bazą danych Oracle:

```
spring.datasource.url= jdbc:oracle:thin:@localhost:1521:xe  
spring.datasource.username=system  
spring.datasource.password=admin123  
spring.datasource.driver-class-name=oracle.jdbc.OracleDriver  
  
spring.jpa.hibernate.ddl-auto=update  
server.port=8081
```

Umieszczamy adres url naszej bazy, jej port oraz nazwę. Podajemy login i hasło użytkownika, oraz typ „drivera” **oracle.jdbc.OracleDriver**. Następnie ustawiamy auto update’owanie **schema** bazy, gdyż w samym springu też zostały zmieniane niektóre kwestie, np. dodanie sekwencji, tabel pośredniczących w relacjach **wiele do wielu**. Na samym końcu zmieniamy port serwera na **8081**, ponieważ usługi bazy danych Oracle działają na domyślnym porcie **8080**.

W projekcie wykorzystany został model **MVC** (model, view, controller) z racji tego nastąpił podział na odpowiednie katalogi i klasy odpowiadające za poszczególne warstwy modelu. W katalogu „**model**” utworzone zostały encje, w katalogu „**service**” logika biznesowa, „**repository**” operacje dostępu do bazy danych, „**controller**” odbieranie zapytań HTTP i zwracanie odpowiednich stanów zasobów (w tym projekcie w formacie JSON).

Przykładowa encja dla tabeli grupa_zajeciowa, każda taka klasa musi zawierać adnotację **Entity** oraz **ID** i strategię generowania wartości id, w tym przypadku utworzona zostaje sekwencja zwiększająca o 1 wartość id. Każde pole musi mieć getter i seter, musi być konstruktor domyślny jak i z parametrami oraz dodatkowo powinna się znaleźć nadpisana metoda **toString()** oraz metoda **hashująca**.

```
@Entity
public class GrupaZajeciowa {

    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator =
"grupa_zajeciowa_seq")
    @SequenceGenerator(name = "grupa_zajeciowa_seq", sequenceName =
"grupa_zajeciowa_seq", initialValue = 1, allocationSize = 1)
    private Long id_grupy_zajeciowej;
    private String nazwa_grupy_zajeciowej;

    @ManyToMany(mappedBy = "grupyZajeciowe")
    Set<Student> studenci;

    @ManyToMany(mappedBy = "grupyZajeciowe")
    Set<Pracownik> pracownicy;

    @ManyToMany(mappedBy = "grupyZajeciowe")
    Set<Przedmiot> przedmioty;

    public Long getId_grupy_zajeciowej() {
        return id_grupy_zajeciowej;
    }
    public void setId_grupy_zajeciowej(Long id_grupy_zajeciowej) {
        this.id_grupy_zajeciowej = id_grupy_zajeciowej;
    }

    public void setNazwa_grupy_zajeciowej(String nazwa_grupy_zajeciowej)
{
        this.nazwa_grupy_zajeciowej = nazwa_grupy_zajeciowej;
    }

    public String getNazwa_grupy_zajeciowej() {
        return nazwa_grupy_zajeciowej;
    }

    public GrupaZajeciowa() {
    }

    public GrupaZajeciowa(String nazwa_grupy_zajeciowej) {
        this.nazwa_grupy_zajeciowej = nazwa_grupy_zajeciowej;
    }

    @Override
    public String toString() {
        return "GrupaZajeciowa{" +
            "id_grupy_zajeciowej=" + id_grupy_zajeciowej +
            ", nazwa_grupy_zajeciowej='" + nazwa_grupy_zajeciowej +
            '\'' +
            '\'';
        }
    }
}
```

Adnotacja **@ManyToMany** musi zostać użyta w obu klasach połączonych relacją:

```
@ManyToMany(mappedBy = "grupyZajeciowe")
private Set<Student> studenci;
```

Rysunek 1. Klasa GrupaZajeciowa

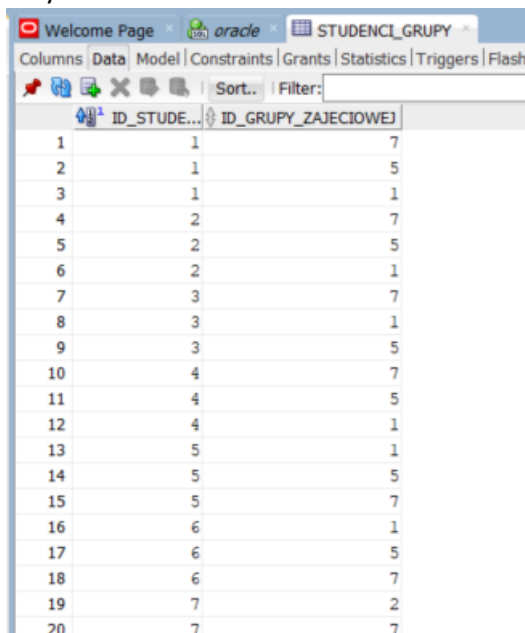
W **Set<Student>** znajdują się wszyscy studenci z danej grupy, w **Set<GrupaZajeciowa>** znajdują się wszystkie grupy danego studenta.

```
@ManyToMany
@JoinTable(
    name = "studenci_grupy",
    joinColumns = @JoinColumn(name = "id_studenta"),
    inverseJoinColumns = @JoinColumn(name =
    "id_grupy_zajeciowej"))
private Set<GrupaZajeciowa> grupyZajeciowe;
```

Rysunek 2. Klasa Student

@JoinTable pozwala na połączenie obu tablic w tablicy pomocniczej o **name = „studenci_grupy”**, z nowo utworzoną tabelą występuje relacja na podstawie **id_studenta** z tabelą **student** i na podstawie **id_grupy_zajeciowej** z tabelą **grupa_zajeciowa**.

Przykładowa zawartość tabeli:



The screenshot shows the Oracle SQL Developer interface with the 'STUDENCI_GRPY' table selected. The table has two columns: 'ID_STUDE...' and 'ID_GRPY_ZAJECIOWEJ'. The data is displayed in a grid with 20 rows.

	ID_STUDE...	ID_GRPY_ZAJECIOWEJ
1	1	7
2	1	5
3	1	1
4	2	7
5	2	5
6	2	1
7	3	7
8	3	1
9	3	5
10	4	7
11	4	5
12	4	1
13	5	1
14	5	5
15	5	7
16	6	1
17	6	5
18	6	7
19	7	2
20	7	7

Przykładowe repository **StudentRepository**, rozszerza interfejs **JpaRepository**<typ klasy, do której piszemy repository, typ pola ID tej klasy>

```
public interface StudentRepository extends JpaRepository<Student, Long>
{

    @Query("SELECT s FROM Student s WHERE s.imie_studenta = ?1 AND
s.nazwisko_studenta = ?2")
    Optional<Student>
    findStudentByImie_studentaAndNazwisko_studenta(@Param("imie") String
    imie, @Param("nazwisko") String nazwisko);

    @Modifying
    @Transactional
    @Query("update Student s set s.imie_studenta = :imie,
s.nazwisko_studenta = :nazwisko, " +
        "s.indeks_studenta = :indeks where s.id_studenta = :id")
    void update(@Param("imie") String imie, @Param("nazwisko") String
    nazwisko,
        @Param("indeks") String indeks, @Param("id") Long id);

    @Procedure("PROCEDURE_ADD_GRUPA_STUDENT")
    void addGrupaZajeciowa(Long id_studenta, Long id_grupy_zajeciowej);

    @Procedure(name = "Student.getOceny")
    ResultSet getOceny(@Param("id") Long id);
}
```

Interfejs ma wiele już napisanych metod, które możemy wywołać, jak **findById(Long id)**, **findAll()** itd. Możemy również napisać własne metody dzięki użyciu **@Query**(„treść zapytanie). Aby użyć składni np.: PL/SQL należy ustawić parametr **nativeQuery = true**. Zapytania modyfikujące dane należy oznaczyć adnotacjami **@Modifying**, **@Transactional**. Aby obsłużyć procedury należy użyć adnotacji **@Procedure(nazwa_procedury)** i zapisać dokładne nazwy jej parametrów i ich typy na liście pobieranych zmiennych w metodzie.

```
@Procedure("PROCEDURE_ADD_GRUPA_STUDENT")
void addGrupaZajeciowa(Long id_studenta, Long id_grupy_zajeciowej);
```

```
create or replace PROCEDURE PROCEDURE_ADD_GRUPA_STUDENT(id_studenta IN NUMBER, id_grupy_zajeciowej IN NUMBER)
```

StudentService tutaj wykonywana jest praktycznie cała logika sprawdzania poprawności danych, występowania ich w bazie, pobierania z bazy, umieszczania, zmieniania itd.

Metody pobierają cały zbiór studentów, lub studenta o szczególnym id.

```
public List<Student> getAll() {  
    return studentRepository.findAll();  
}  
  
public Optional<Student> getStudentById(Long id) {  
    Optional<Student> studentObj = Optional.empty();  
    if (id != null && id > 0) {  
        studentObj = studentRepository.findById(id);  
    }  
  
    return studentObj;  
}
```

Dodawanie studenta do bazy, sprawdzone zostają odpowiednie wartości umieszczanego do bazy studenta oraz czy już nie istnieje taki w bazie, na końcu zostaje wykorzystana metoda **save()** z **studentRepository** i zwrócony zostaje nowo utworzony Student. Gdy student istnieje zwrócony zostaje null. (nie udało się dodać do bazy).

```
public Student save(Student student) {  
    Student studentObj = null;  
  
    if (student.getImie_studenta() != null &&  
        !student.getImie_studenta().equals("")) {  
        if (student.getNazwisko_studenta() != null &&  
            !student.getNazwisko_studenta().equals("")) {  
            Optional<Student> studentRep =  
studentRepository.findStudentByImie_studentaAndNazwisko_studenta(  
                student.getImie_studenta(),  
                student.getNazwisko_studenta()  
            );  
            if (studentRep.isEmpty()) { //nie ma w bazie, dodajemy  
                studentObj = studentRepository.save(new Student(  
                    student.getImie_studenta(),  
                    student.getNazwisko_studenta(),  
                    student.getIndeks_studenta()  
                ));  
            }  
        }  
    }  
  
    return studentObj;  
}
```

Usuwanie studenta o danym id, po sprawdzeniu istnienia danego studenta zostaje wykorzystana domyślna metoda odziedziczona z JpaRepository o nazwie **deleteById(Long id)**. Zwrócony zostaje usunięty obiekt lub null gdy nie uda się go usunąć.

```
public Optional<Student> delete(Long id) {
    Optional<Student> studentObj = Optional.empty();
    if (id != null && id > 0) {
        if (studentRepository.existsById(id)) {
            //istnieje
            studentObj = studentRepository.findById(id);
            studentRepository.deleteById(id);
        }
    }

    return studentObj;
}
```

Update'owanie danego studenta, po sprawdzeniu szeregu warunków gdy uda nam się zmienić wartości w jego kolumnach zwrócony zostaje kod **200 (OK)** oraz stary student przed zmianą.

Nieudane update'owanie zwróci kod **406(NOT_ACCEPTABLE)**

```
public ResponseEntity<Student> update(Student student, Long id) {
    Student studentObj = null;
    if (id != null && id > 0) {
        if (studentRepository.existsById(id)) {
            //istnieje
            studentObj = studentRepository.findById(id).get();
            studentRepository.update(student.getImie_studenta(),
                                     student.getNazwisko_studenta(),
                                     student.getIndeks_studenta(),
                                     id);
            return new ResponseEntity<>(studentObj, HttpStatus.OK);
        }
        studentObj = this.save(student);
        return new ResponseEntity<>(studentObj, HttpStatus.CREATED);
    }

    return new ResponseEntity<>(HttpStatus.NOT_ACCEPTABLE);
}
```

Przykładowa obsługa procedury, dla podanych wartości id studenta i grupy i sprawdzeniu szeregu warunków istnienia ich w bazie wywołana zostaje procedura

studentRepository.addGrupaZajeciowa(student_id, grupa_id);

Gdy wykona się poprawnie zwrócony zostaje kod **200** oraz grupa do której student został dodany. W przeciwnym wypadku zwrócony tylko zostaje kod **406**.

```
public ResponseEntity<?> addGupaZajeciowa(Long student_id, Long
grupa_id) {
    Optional<GrupaZajeciowa> grupaZajeciowaObj = Optional.empty();
    if (grupa_id != null && grupa_id > 0) {
        if (grupaZajeciowaRepository.existsById(grupa_id)) {
            if (student_id != null && student_id > 0) {
                if (studentRepository.existsById(student_id)) {
                    grupaZajeciowaObj =
grupaZajeciowaRepository.findById(grupa_id);
                    studentRepository.addGrupaZajeciowa(student_id,
grupa_id);
                    return new ResponseEntity<>(grupaZajeciowaObj,
HttpStatus.OK);
                }
            }
        }
    }

    return new ResponseEntity<>(HttpStatus.NOT_ACCEPTABLE);
}
```

StudentController dla odpowiednich zapytań http wystanych z odpowiednim ciałem i parametrami w zapytaniu na odpowiedni adres wywołane zostają metody z poprzednio opisanego **StudentService**.

```
@RestController

@RequestMapping("/api/studenci")
public class StudentController {

    private final StudentService studentService;

    @Autowired
    public StudentController(StudentService studentService) {
        this.studentService = studentService;
    }

    @GetMapping
    public List<Student> getAll() {
        return studentService.getAll();
    }

    @GetMapping("/{id}")
    public Optional<Student> getById(@PathVariable Long id) {
        return studentService.getStudentById(id);
    }

    @PostMapping
    public Student saveStudent(@RequestBody Student student) {
        return studentService.save(student);
    }

    @DeleteMapping("/{id}")
    public Optional<Student> delete(@PathVariable Long id) {
        return studentService.delete(id);
    }

    @PatchMapping("/{id}")
    public ResponseEntity<Student> update(@PathVariable Long id,
    @RequestBody Student student) {
        return studentService.update(student, id);
    }

    @GetMapping("/{id}/oceny")
    public ResponseEntity<?> getOcenyById(@PathVariable Long id) throws
    SQLException, IOException {
        return studentService.getOceny(id);
    }

    @PostMapping("/{id_studenta}/grupy-zajeciowe/{id_grupy_zajeciowej}")
    public ResponseEntity<?> addGrupaZajeciowa(@PathVariable Long
    id_studenta, @PathVariable Long id_grupy_zajeciowej) {
        return studentService.addGupaZajeciowa(id_studenta,
    id_grupy_zajeciowej);
    }
}
```

Dodatkowa metoda wykorzystująca procedurę pobierającą oceny dla danego studenta. Rozwiązanie problemu polegającego na poprawnym skonfigurowaniu zarówno procedury by zwracała zbiór wierszy (u mnie zwraca kursor) jak i zmapowania ich na format **JSON** zajęło mi najwięcej czasu. Metoda tworzy **zmienną typu ResultSet** do niej zwracany jest wynik działania procedury, później pobierane są metadane z wynikowego zbioru, takie jak ilość kolumn czy nazwa kolumn. W pętli następuje mapowanie wierszy do Mapy<String, Object>, wiersze umieszczane zostają w liście, cała lista wraz z kodem http.OK **200** zostaje po pomyślnym wykonaniu zwrócona do controllera, a tam do użytkownika wysyłającego na odpowiedni **end-point** odpowiednie **zapytanie**.

```
public ResponseEntity<?> getOceny(Long id) throws SQLException,
IOException {

    ResultSet resultSet = studentRepository.getOceny(id);
    ResultSetMetaData resultSetMetaData = resultSet.getMetaData();

    //map result set to JSON
    List<Map<String, Object>> rows = new ArrayList<>();
    int numberOfColumns = resultSetMetaData.getColumnCount();

    while (resultSet.next()) {
        Map<String, Object> row = new HashMap<>();
        for (int i = 1; i <= numberOfColumns; i++) {
            String columnName = resultSetMetaData.getColumnName(i);
            //nazwa kolumny
            Object columnValue = resultSet.getObject(i);
            row.put(columnName, columnValue);
        }
        rows.add(row);
    }

    return new ResponseEntity<>(rows, HttpStatus.OK);
}
```

Zapytanie hierarchiczne

```
WITH cte AS
(
SELECT  s.nazwisko_studenta AS nazwisko, s.id_studenta AS id_,
pr1.id_pracownika + 100 AS id_2
FROM    ocena o
INNER JOIN student s
ON      s.id_studenta = o.id_studenta
INNER JOIN przedmiot p
ON      o.id_przedmiotu = p.id_przedmiotu
INNER JOIN pracownik pr1
ON      pr1.id_przedmiotu = p.id_przedmiotu
UNION ALL
SELECT  pr2.nazwisko_pracownika AS nazwisko, pr2.id_pracownika + 100 AS
id_, null AS id_2
FROM    pracownik pr2
) SELECT lpad(' ', 4*(level-1)) || nazwisko, id_, id_2, level
FROM    cte
start with id_2 is null
```

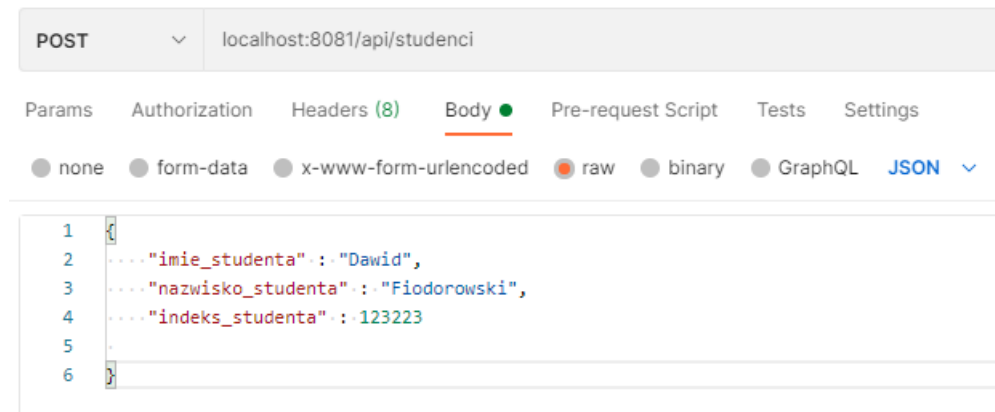
Z racji przymusu połączenia wielu tabel w 1 i zabawy nad takim ustawieniem i przypisaniem wartości rozróżniających studenta od pracownika było to jedno z trudniejszych zadań. Każdy pracownik ma dopisane + 100 do swojego, tak aby dało się go odróżnić od studenta, dodatkowo po użyciu UNION ustawiono nazwę id_studenta jak i zmienionego id_pracownika na **id_**, id pracownika wystawiającego daną ocenę dla studenta nazwano **id_2**, pracownik ma tam wpisaną wartość **null**.

Przykładowy wynik:

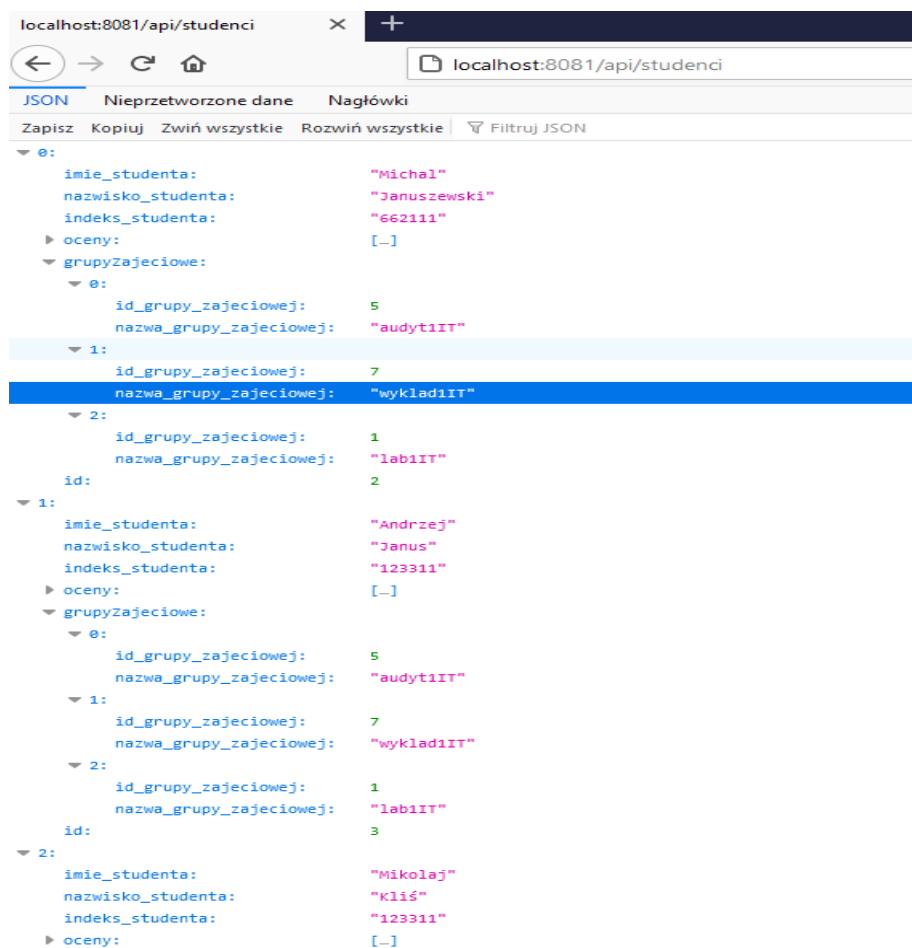
	LPAD(' ', 4*(LEVEL-1)) NAZWISKO	ID_	ID_2	LEVEL
1	Michalowski	102	(null)	1
2	Kowalski	1	102	2
3	Januszeowski	2	102	2
4	Januszeowski	2	102	2
5	Januszeowski	2	102	2
6	Janus	3	102	2
7	Kliś	4	102	2
8	Żbik	5	102	2
9	Ryś	6	102	2
10	Mikołajewski	9	102	2
11	Krzyk	11	102	2
12	Kowalski	13	102	2
13	Kowalewicz	14	102	2
14	Dubiel	17	102	2
15	Jarukiewicz	21	102	2
16	Zbyszeowski	103	(null)	1
17	Kowalski	1	103	2
18	Kowalski	1	103	2
19	Janus	3	103	2
20	Glik	12	103	2
21	Glik	12	103	2
22	Kowalski	13	103	2
23	Kowalewicz	14	103	2
24	Dubiel	17	103	2
25	Kruk	20	103	2
26	Jarukiewicz	21	103	2
27	Bykowski	22	103	2
28	Zabielski	23	103	2
29	Michniewicz	24	103	2
30	Lew	104	(null)	1
31	Januszeowski	2	104	2
32	Januszeowski	2	104	2
33	Januszeowski	2	104	2

Przykłady działania

Ciało zapytania aby stworzyć użytkownika:



Wypisanie listy studentów:



Wypisanie informacji o studencie o id 5

localhost:8081/api/studenci/5

JSON Nieprzetworzone dane Nagłówki

Zapisz Kopiuj Zwiń wszystkie Rozwiń wszystkie Filtruj JSON

```
{  "imie_studenta": "Mikolaj",  "nazwisko_studenta": "Żbik",  "indeks_studenta": "871233",  "oceny": [    {      "id_oceny": 4,      "wartosc": "3",      "waga_oceny": 1,      "przedmiot": {        "id_przedmiotu": 1,        "nazwa_przedmiotu": "Algebra"      }    },    {      "id_oceny": 93,      "wartosc": "4",      "waga_oceny": 3,      "przedmiot": {        "id_przedmiotu": 7,        "nazwa_przedmiotu": "Analiza matematyczna"      }    }  ],  "grupyZajeciowe": [    {      "id_grupy_zajeciowej": 5,      "nazwa_grupy_zajeciowej": "audyt1IT"    },    {      "id_grupy_zajeciowej": 1,      "nazwa_grupy_zajeciowej": "lab1IT"    },    {      "id_grupy_zajeciowej": 7,      "nazwa_grupy_zajeciowej": "wyklad1IT"    }  ],  "id": 5}
```

Update danych studenta o id 1:

PATCH localhost:8081/api/studenci/1 Send

Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify

```
1 { 2   "imie_studenta": "Dawid", 3   "nazwisko_studenta": "Fiodorowski", 4   "indeks_studenta": "123223" 5 } 6 }
```

Delete studenta o danym id:

DELETE localhost:8081/api/studenci/1

Podsumowanie

Współpraca z bazami Oracle za pomocą SQLDevelopera była dosyć przyjemną czynnością dodatkowo, połączenie się z bazą w Spring'u polegało na zapisaniu +/- 10 linii kodu. Dosyć czasochłonną czynnością okazało się jednak tworzenie wszystkich encji oraz controllerów i pisanie kodu sprawdzającego poprawność danych wpisywanych do tabeli. Projekt nauczył mnie wielu rzeczy i pomógł w ugruntowaniu poznanej już wiedzy. Początki pracy z bazą i instalacja okazały się zbyt problematyczne, dlatego zainstalowana została przestarzała wersja **11g**.

Projektowi przyda się gruntowny refactor, ze względu na mieszanie nazw angielskich z polskimi dlatego iż zacząłem projektowanie od utworzenia bazy w języku Polskim, gdzie wszystkie metody w Springu robiłem przy użyciu angielskiego słownictwa. W planach dodatkowo jest utworzenie front-endu i poprawa controllerów aby zwracały odpowiednie kody statusu http itd.

Link do repozytorium github: - <https://github.com/Radglay/wirtualny-dziekanat>