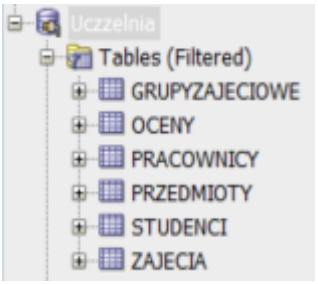
Wydział WIMiIP	Imię i nazwisko 1.Kacper Bielak 2.Tomasz Artecki 3.Dominik Budzowski	Rok 2	Grupa GL01	
Temat: [CWP1] Projekt Oracle				
Data wykonania 25.05.2021	Przedmiot: Bazy danych			OCENA

1. Idea projektu

Baza danych służąca do zarządzania dziekanatem. Zawiera tabele grup zajęciowych, ocen, pracowników, przedmiotów, studentów oraz zajęć. Umożliwia dodawanie i usuwanie rekordów za pomocą procedur. Użytkownik może z łatwością selekcjonować oraz wyświetlać najpotrzebniejsze dane za pomocą widoków i innych narzędzi. Projekt bazy jest rozwinięciem projektu bazy danych z Oracla przygotowywanego na kilku poprzednich zajęciach.



2. Diagramy z Oracle Data Modeler (najlepiej diagram początkowy a jak podczas pracy nad projektem nastąpią jakieś zmiany proszę przedstawić oba w celu porównania)

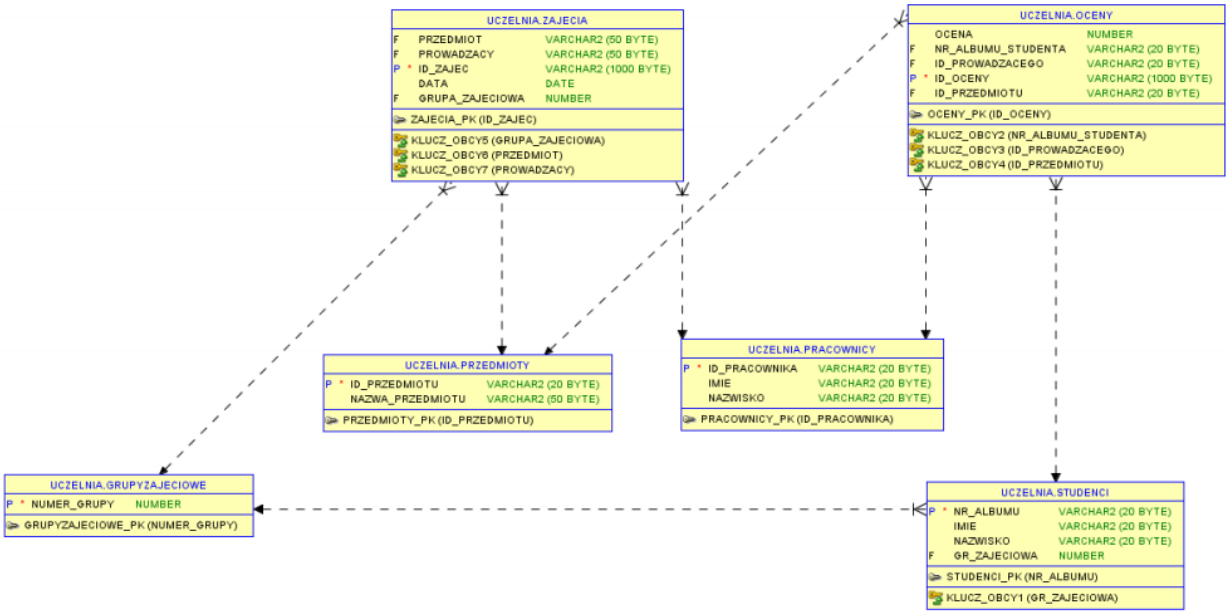


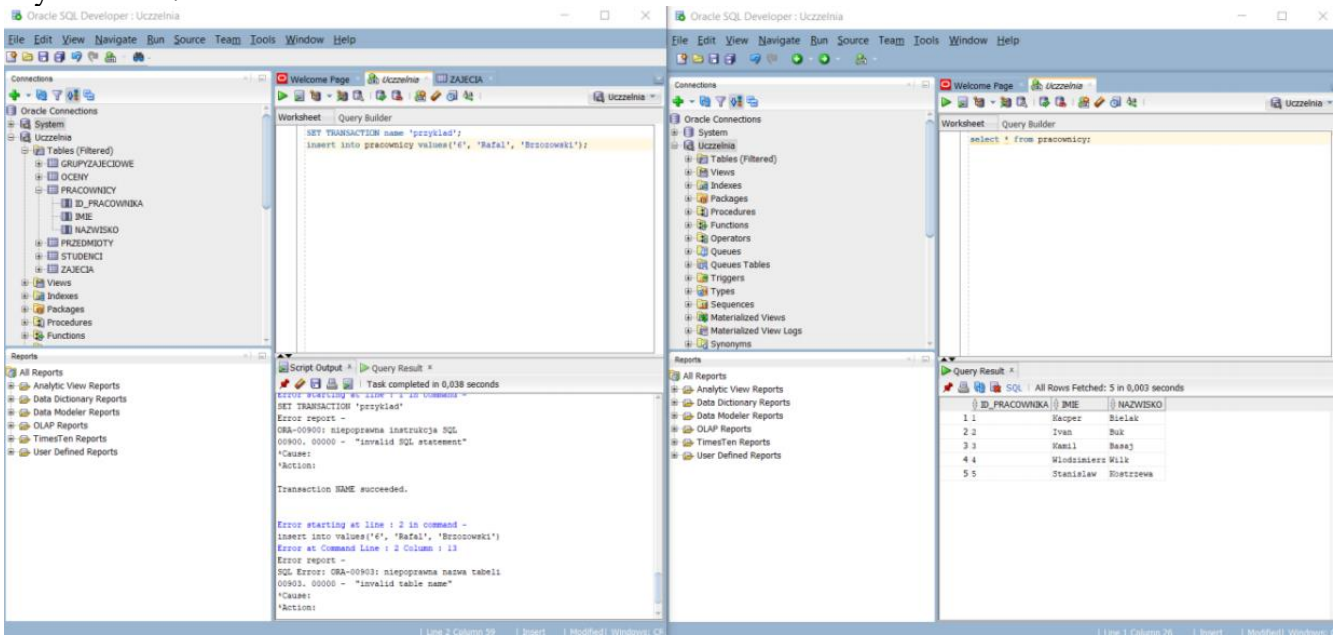
Tabela zajecia powiązana jest z tabelą grupyzajeciowe poprzez numer_grupy, przedmioty poprzez id_przedmiotu oraz z pracownicy poprzez id_pracownika. Tabela oceny powiązana jest z tabelą studenci poprzez numer_alubumu_studenta, pracownicy poprzez id_pracownika oraz z przedmioty poprzez id_przedmiotu. Tabela studenci powiązana jest z tabelą grupyzajeciowe poprzez numer_grupy.

Zmiany jakie nastąpiły względem początkowych relacji między tabeli były takie, że tabela przedmioty zawierała grupę zajęciową i była z tą tabelą(grupyzajeciowe) powiązana. Jednakże przy wstawianiu rekordów występowały liczne błędy dlatego zrezygnowano z tego powiązania i zamiast niego powiązano tabele zajęcia z tą tabelą grupyzajeciowe poprzez wstawienie tam kolumny grupy_zajeciowe i dodanie klucza obcego wiążącego tą tabele z tabelą grupyzajeciowe.

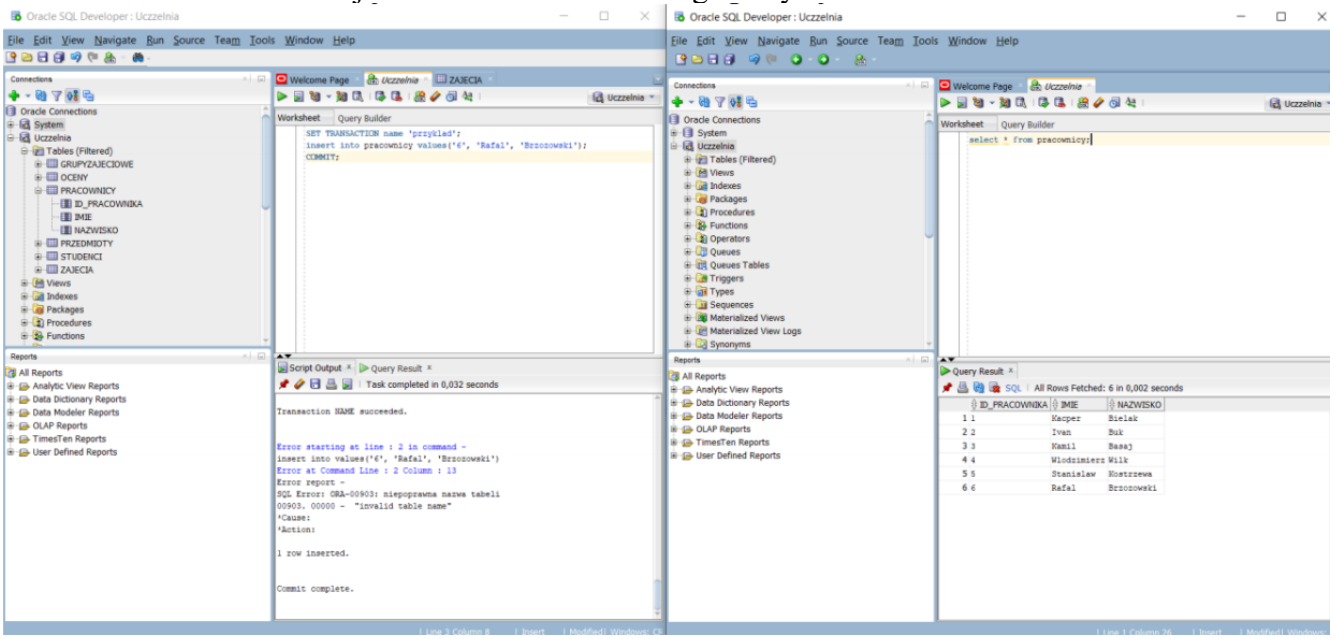
3. **SQL DCL – utworzenie użytkowników i nadanie im uprawnień**
   W bazie utworzono 2 użytkowników, jednym z nich jest dziekanat, który ma wszystkie uprawnienia do całej bazy, a drugim jest pracownik, które może operować tylko na tabeli *oceny*:

```
create user c##dziekanat identified by dz;
create user c##pracownik identified by pr;
grant all privileges to c##dziekanat;
grant all PRIVILEGES on oceny to c##pracownik;
grant create session to c##pracownik;
```

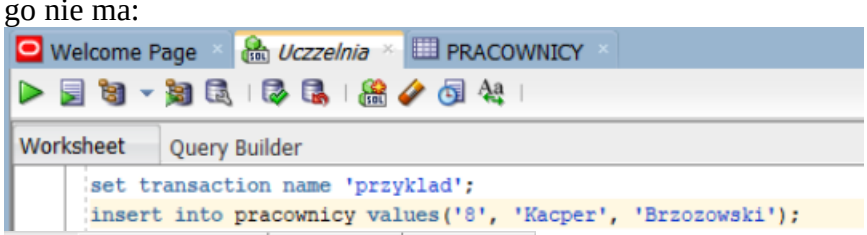
4. **Wykorzystanie transakcji i pokazanie że działa**
   Dodanie nowego pracownika bez zakończenia transakcji. Widoczny brak efektu na drugim użytkowniku:



Po zakomitowaniu transakcji widoczna zmiana na drugim użytkowniku:



Przykład z rollbackiem – początkowo widać nowo dodanego pracownika, po odrzuceniu zmian już go nie ma:



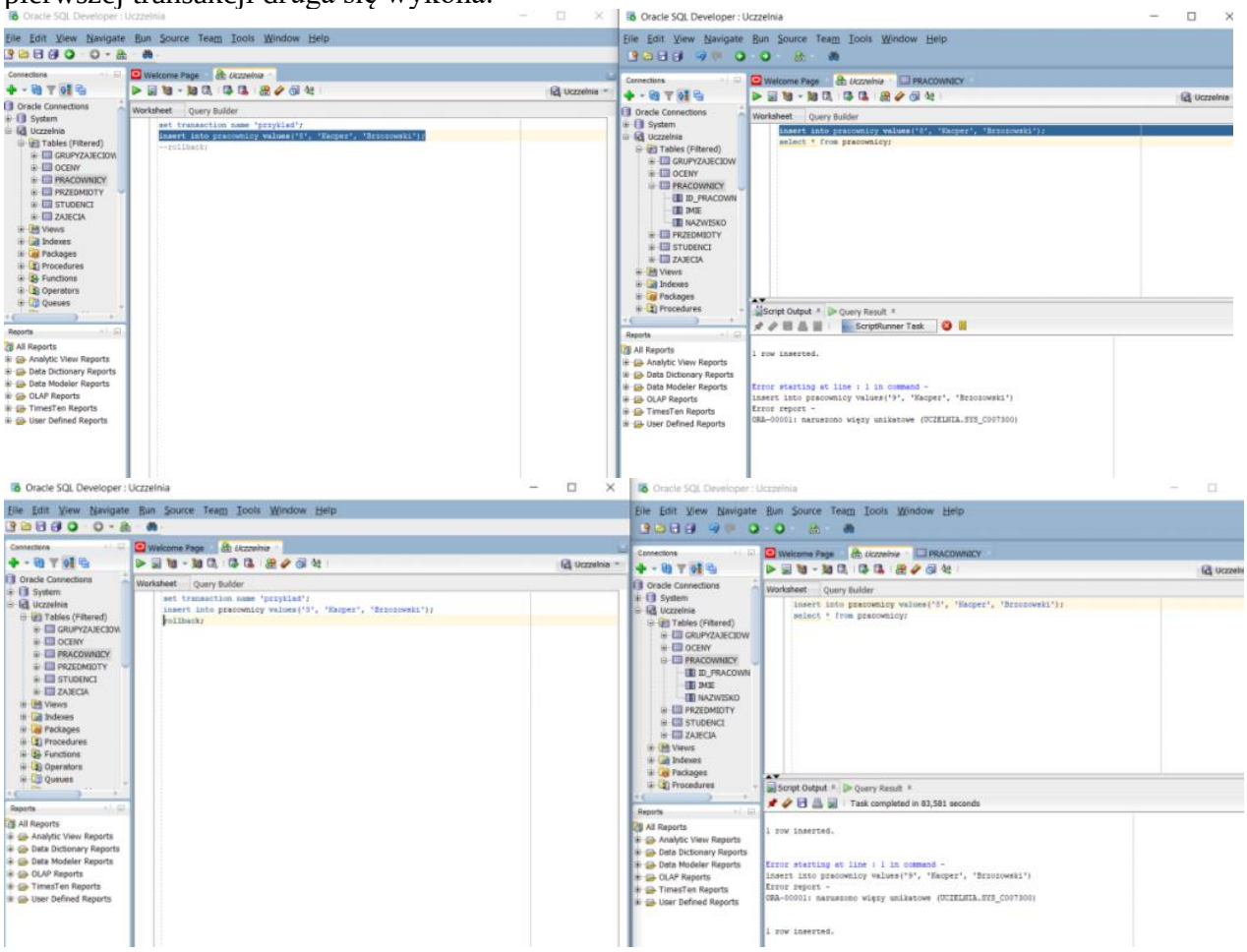
	ID_PRACOWNIKA	IMIE	NAZWISKO
1	1	Kacper	Bielak
2	2	Ivan	Buk
3	3	Kamil	Basaj
4	4	Wlodzimierz	Wilk
5	5	Stanislaw	Kostrzewa
6	6	Rafal	Brzozowski
7	7	Krzysztof	Brzozowski
8	8	Kacper	Brzozowski

Przed rollbackiem

	ID_PRACOWNIKA	IMIE	NAZWISKO
1	1	Kacper	Bielak
2	2	Ivan	Buk
3	3	Kamil	Basaj
4	4	Wlodzimierz	Wilk
5	5	Stanislaw	Kostrzewa
6	6	Rafal	Brzozowski
7	7	Krzysztof	Brzozowski

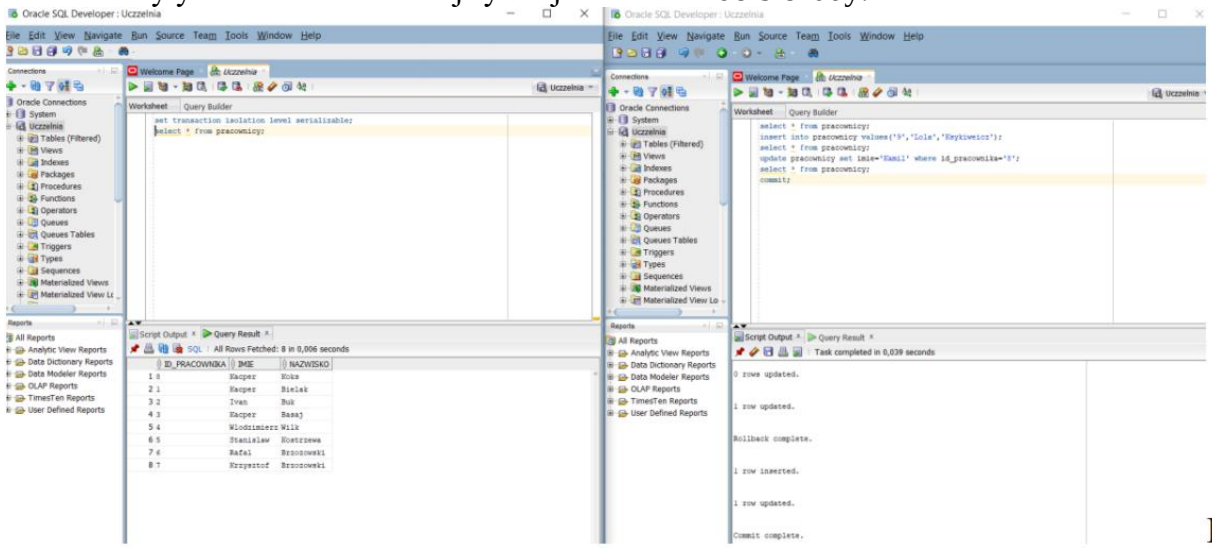
Po rollbacku

Zablokowanie transakcji w przypadku gdy na jednym użytkowniku jest odpalona transakcja operująca na tym samym wierszu co transakcja u innego użytkownika, dopiero po zakończeniu pierwszej transakcji druga się wykona:

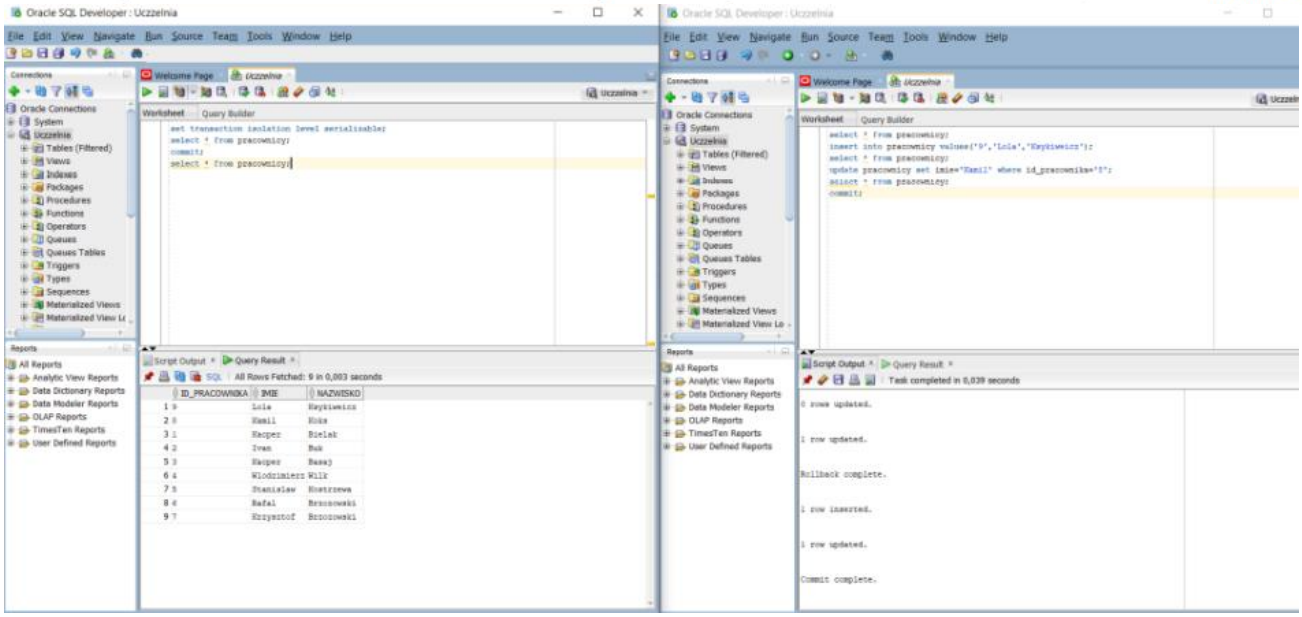


W celu unikania zakleszczeń stosuje się różne **poziomy izolacji transakcji** w Oracle. Poziom izolacji transakcji jest stopniem, do jakiego zmiany dokonywane przez jedną transakcję są od dzielone od innej transakcji wykonywanej współbieżnie. Do tych poziomów izolacji należą: READ UNCOMMITTED, READ COMMITED, REPEATABLE READ, SERIALIZABLE.

Przykład dla poziomu serializable:
Ustawiono tryb izolacji na serializable. Na jednym użytkowniku wypisano rekordy, na drugim dodano rekord do tej tablicy i zmodyfikowano istniejący po czym zakomitowano zmiany. Następnie na pierwszym użytkowniku wypisano rekordy i te zmiany wprowadzone na drugim użytkowniku nie zostały wyświetlone. Dopiero po zakomitowaniu zmian na pierwszym użytkowniku i wyświetleniu rekordów były one widoczne. W tej sytuacji nie doszło do blokady:



Brak zmian

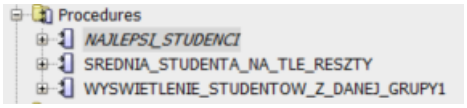


Zmiany widoczne po zakomitowaniu

5. Procedury, Wyzwalacze, Widoki

a) Procedury

Projekt zawiera 3 procedury, które odpowiadają za wyświetlenie danej liczby najlepszych studentów w rankingu, średniej danego studenta wraz z średnia całego kierunku oraz za wyświetlenie studentów z danej grupy.



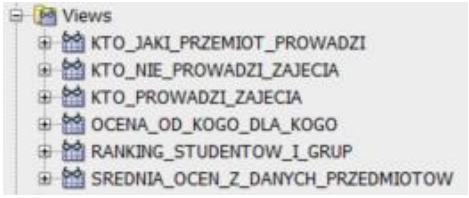
b) Wyzwalacze

Projekt zawiera 2 wyzwalacze. Jeden z nich odpowiadał za brak możliwości zmiany oceny na niższą(before), a drugi za wyświetlenie odpowiedniego komunikatu gdy użytkownik dostał ocenę niedostateczną.

c) Widoki

Projekt zawiera 6 widoków:

- ***kto_jaki_przedmiot_prowadzi*** - wyświetla informacje o prowadzących dany przedmiot;
- ***kto_nie_prowadzi_zajec*** - wyświetla informacje o pracownikach, którzy nie prowadzą zajęć;
- ***kto_prowadzi_zajecia*** - wyświetla informacje o pracownikach, którzy prowadzą zajęć;
- ***ocena_od_kogo_dla_kogo*** – wyświetla dane studenta, prowadzącego, który wystawił daną ocenę, ocenę i id_oceny;
- ***ranking_studentow_i_grup*** – wyświetla ranking studentów względem średniej oraz średnią poszczególnych grup;
- ***srednia_ocen_z_danych_przedmiotow*** - wyświetla średnie ocen z danego przedmiotu.



6. Złączenia:

1. **MINUS**

   Złączenie to zostało wykorzystane w widoku kto_nie_prowadzi_zajec. Kod ten znajduje się poniżej:

```
create view kto_nie_prowadzi_zajecia as
select imie, nazwisko, id_pracownika from pracownicy where id_pracownika IN
(select id_pracownika from pracownicy minus select prowadzacy from zajecia)
order by nazwisko, imie;
```

2. **INTERSECT**

   Złączenie to zostało wykorzystane w widoku kto_prowadzi_zajecia. Kod ten znajduje się poniżej:

```
create view kto_prowadzi_zajecia as
select * from pracownicy where id_pracownika in
(select id_pracownika from pracownicy
intersect select prowadzacy from zajecia);
```

3. **SQL JOIN**

   Złączenia te były wykorzystywane wiele razy w projekcie. Poniżej znajduje się jeden z przykładów:

```
create view ranking_studentow_i_grup as
select studenci.nazwisko,studenci.gr_zajeciowa, avg(oceny.ocena) as
srednia,rank() over (order by avg(oceny.ocena) DESC) as rank
from oceny left join studenci on studenci.nr_albumu=oceny.nr_albumu_studenta
group by studenci.gr_zajeciowa, cube(studenci.nazwisko) order by rank,
studenci.gr_zajeciowa;
```

7. Zapytania o określoną liczbę wierszy z wykorzystywaniem (Fetch First, Offset)

1. **Fetch First**
   Funkcja ta została wykorzystana w różnych miejscach, m.in. przy tworzeniu procedury najlepsi_studenci. Dzięki niej wyświetlana jest taka liczba najlepszych studentów jaka jest podana przy wywoływaniu procedury. Kod ten znajduje się poniżej:

```
create PROCEDURE najlepsi_studenci(liczba IN number)
as c1 sys_refcursor;
BEGIN
open c1 for
select * from ranking_studentow_i_grup where is not null
fetch first liczba rows only;
dbms_sql.return_result(c1);
END;
```

2. **Offset**
   Funkcja ta została wykorzystana przy tworzeniu procedury wyświetlenie_studentow_z_danej_grupy. Dzięki niej zostają wyświetlenie studenci tylko z danej grupy. Kod przedstawiający to znajduje się poniżej:

```
create PROCEDURE wyswietlenie_studentow_z_danej_grupy1( liczba IN number ) as
c1 sys_refcursor;
BEGIN
open c1 for
select imie, nazwisko, nr_albumu from studenci where gr_zajeciowa=(select
numer_grupy from grupyzajeciowe offset liczba-1 rows fetch next 1 rows only);
dbms_sql.return_result(c1);
END;
```

8. 2-3 przypadki użycia funkcji związanych z Analizą Danych (np. PIVOT, MODEL)

1. **PIVOT**
   Funkcja ta została wykorzystana przy tworzeniu widoku srednia_ocen_z_danych_przedmiotow. Dzięki niej zostaje wyświetlona średnia ocen z danego przedmiotu, gdzie kolumny są przedmiotami. Kod ten znajduje się poniżej:

```
create view srednia_ocen_z_danych_przedmiotow as
select * from (select id_przedmiotu, ocena from oceny) pivot(avg(ocena) AS
SREDNIA for id_przedmiotu IN (1 as ANALIZA,2 AS ALGEBRA,3 AS BAZY_DANYCH,4 AS
PODSTAWY_PROGRAMOWANIE,5 AS PODSTAWY_PROGRAMOWANIE_OBIEKTOWE));
```

2. **RANK**
   Funkcja ta została wykorzystana wielokrotnie w naszym projekcie, m.in. w widoku ranking_studentow_i_grup. Dzięki niej średnie ocen zostały przedstawione w formie rankingu. Kod z tą funkcją znajduję się poniżej:

```
create view ranking_studentow_i_grup as select
studenci.nazwisko,studenci.gr_zajeciowa, avg(oceny.ocena) as srednia,rank()
over (order by avg(oceny.ocena) DESC) as rank
from oceny left join studenci on studenci.nr_albumu=oceny.nr_albumu_studenta
group by studenci.gr_zajeciowa, cube(studenci.nazwisko) order by rank,
studenci.gr_zajeciowa;
```

9. **Grupowanie z wykorzystaniem ROLLUP i CUBE**

    1. **ROLLUP**
    Funkcja ta została wykorzystana przy pisaniu procedury srednia_studenta_na_tle_reszty. Dzięki niej procedura ta może porównać średnią danego studenta z średnią ocen wszystkich studentów. Kod z tą funkcją znajduje się poniżej:

```
create PROCEDURE srednia_studenta_na_tle_reszty (naz IN varchar2) as
c1 sys_refcursor;
BEGIN
open c1 for
select studenci.nazwisko, avg(oceny.ocena)
from oceny left join studenci on studenci.nr_albumu=oceny.nr_albumu_studenta
group by rollup(studenci.nazwisko) having studenci.nazwisko like naz or
studenci.nazwisko is null;
dbms_sql.return_result(c1);
END;
```

    2. **CUBE**
    Funkcja ta została wykorzystana przy tworzeniu widoku ranking_studentow_i_grup. Dzięki niej była możliwość wyświetlenia średnich ocen dla poszczególnych grup. Kod z tą funkcją znajduję się poniżej:

```
create view ranking_studentow_i_grup as
select studenci.nazwisko,studenci.gr_zajeciowa, avg(oceny.ocena) as
srednia,rank() over (order by avg(oceny.ocena) DESC) as rank
from oceny left join studenci on studenci.nr_albumu=oceny.nr_albumu_studenta
group by studenci.gr_zajeciowa, cube(studenci.nazwisko) order by rank,
studenci.gr_zajeciowa;
```

10. **Jeśli projekt pozwoli - zapytanie hierarchiczne, jeśli nie pozwoli - uzasadnienie dlaczego nie**

Projekt ten nie pozwala na użycie zapytań hierarchicznych, ponieważ w tejże bazie danych nie ma zależności hierarchicznych między rekordami. Projekt – dziekanat nie zawiera takich informacji jak np. to, że dana grupa studentów ma swojego prowadzącego, który natomiast miałby nad sobą rektora. W takiej sytuacji możliwe byłoby użycie tych zapytań. Co więcej, na naszej uczelni grupy nie mają opiekuna więc jest to zbędne.

11. **Próby zaimplementowania kilku "CTE" w Oraclu (wykorzystać wiedzę z MS SQL Server oraz z klauzuli WITH)**

CTE zostało użyte przy tworzeniu widoku kto_jaki_przedmiot_prowadzi. Dzięki niemu kod sql jest bardziej przejrzysty, dane pochodzące z różnych tabel są uporządkowane. Kod tego widoku został przedstawiony poniżej:

```
create view kto_jaki_przemiot_prowadzi as with
cte1 as (select imie, nazwisko, id_pracownika from pracownicy),
cte2 as (select prowadzacy, przedmiot from zajecia),
cte3 as (select id_przedmiotu, nazwa_przedmiotu from przedmioty)
select nazwisko,imie, nazwa_przedmiotu from
cte1 inner join cte2 on cte1.id_pracownika=cte2.prowadzacy
inner join cte3 on cte3.id_przedmiotu=cte2.przedmiot
order by nazwisko;
```

Ponadto CTE zostało wykorzystane przy pisaniu CRUD'a przy wyświetlaniu ocen studentów.

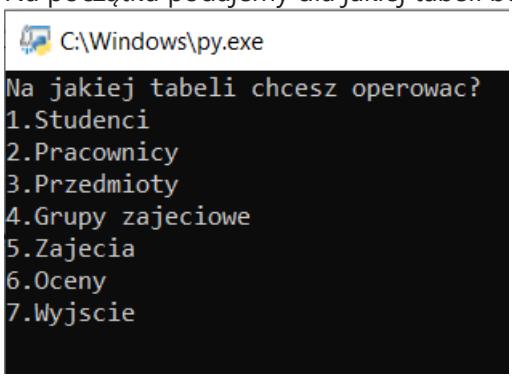
12. CRUD

CRUD został napisany w języku python w środowisku idle korzystając z biblioteki cx_Oracle. Program łączy się z bazą danych Uczelnia i udostępnia przejrzysty interfejs konsolowy do jej zarządzania.

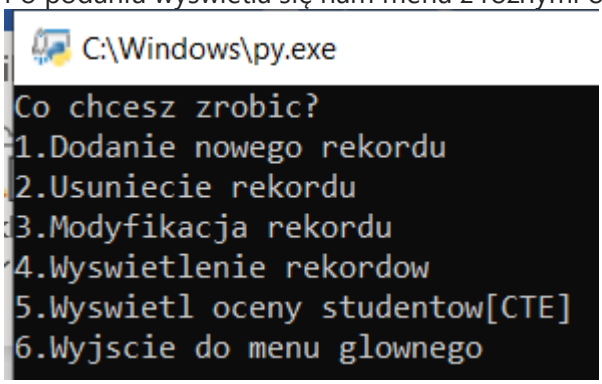
Plik z CRUD'em został dołączony razem ze sprawozdaniem.

Działanie programu

Na początku podajemy dla jakiej tabeli będą wykonywane operacje.

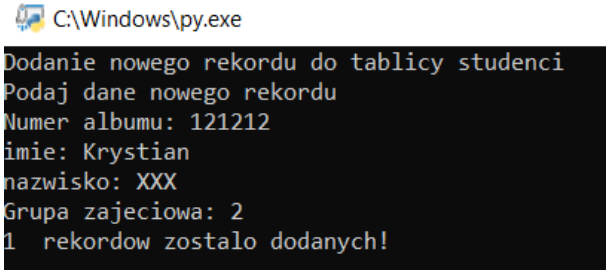


Po podaniu wyświetla się nam menu z różnymi opcjami.



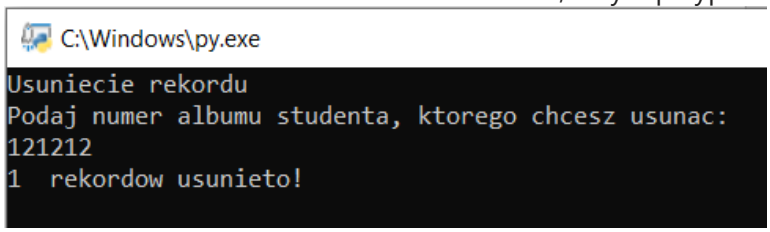
Po wpisaniu odpowiedniej cyfry program wykonuje pewne operacje, a po wykonaniu ich wyświetla ponownie menu, natomiast po wciśnięciu 6 program się zamyka.

1.Dodanie nowego rekordu - dodaje element to tabeli, musimy podać wartości każdej kolumny.

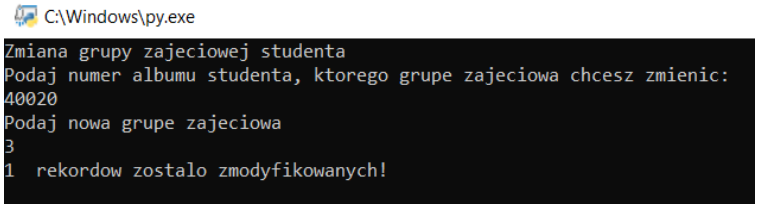


Po wciśnięciu klawisza 'enter' program wraca do poprzedniego menu.

2.Usuniecie rekordu – usuwa rekord z tabeli, w tym przypadku po podaniu numeru albumu studenta.

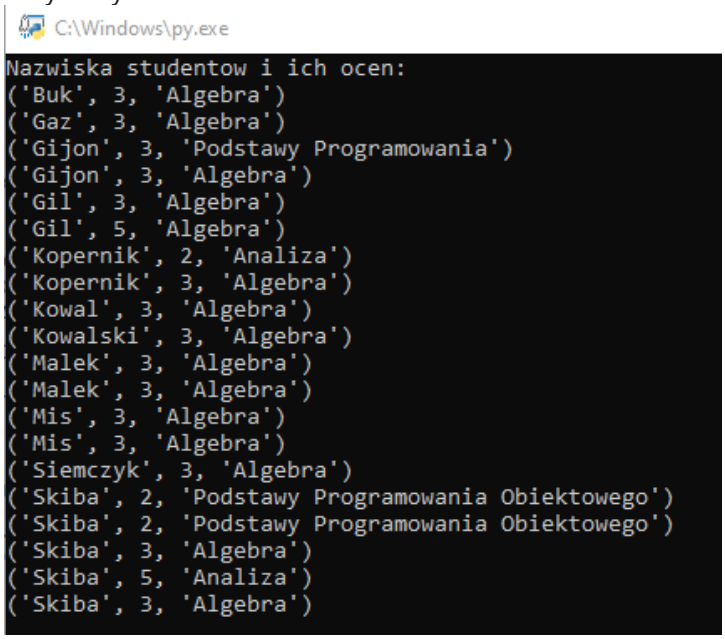


3.Modyfikacja rekordu – zmienia wartość dla jednej z kolumn, w tym przypadku grupę zajęciową.



4. Wyświetl rekordow – wyświetla wszystkie rekordy z danej tablicy.

5. Wyswietl oceny studentow[CTE] – wyświetla oceny poszczególnych studentów z danego przedmiotu z wykorzystaniem CTE.



6. Wyjscie do menu glownego – wyjście do glownego menu

13. Wnioski

Oracle jest środowiskiem bazodanowym bardzo przypominającym MySQL, nie tylko ze względu na relacyjność i język zapytań, ale także graficzny interfejs Oracle'a - SQL Developer. Przypomina on łudząco w obsłudze MySQL Workbencha. Dlatego właśnie tak jak w przypadku GUI MySQL'a - SQL Developer jest bardzo intuicyjnym i wygodnym narzędziem do modelowania baz danych. Bardzo przydatnym programem przy okazji realizacji tego projektu był także SQL Data Modeler, dzięki któremu można było dokładnie rozplanować sobie strukturę bazy tworząc model logiczny, relacyjny i plik DDL, który umożliwiał import szkieletu bazy do Oracle'a. Warto także wspomnieć, że Oracle posiada funkcje, których w MySQL nie znajdziemy jak np. cube, który umożliwia wyświetlenie podsumowań i sum częściowych przy grupowaniu. Jednak nie można także zignorować wad Oracle'a, takich jak: trudne tworzenie użytkowników, ogromna i długa instalacja (z powodu dużej infrastruktury do zainstalowania), a także wyjątkowo nieintuicyjne pierwsze skonfigurowanie tego systemu zarządzania bazami danych. CRUD został napisany w pythonie z powodu łatwego połączenia go z Oracle dzięki istniejącej bibliotece cx_Oracle. Dla większości z nas była to pierwsza styczność z językiem programowania python - okazał się on być bardzo praktycznym językiem programowania ze względu na brak selekcji zmiennych, łatwej do zapamiętania składni i przejrzystego kodu. W porównaniu do poprzednich dwóch środowisk bazodanowych najciężej się nam w nim pracowało.