

Warszawa, 07.06.2021

Politechnika Warszawska  
Wydział Elektroniki i Technik Informacyjnych

Bazy danych 1 (BD1)  
Projekt: Hotel – koordynacja rezerwacjami

Prowadzący: mgr inż. Piotr Maciąg

Wykonawcy:

Krakowiak Aleksandra 290292

Rancew Joanna 300465

## Spis treści

Krótki opis rozwiązania .....	3
Model ER, model relacyjny .....	3
Skrypty do założenia schematu bazy danych.....	5
Skrypty do załadowania danych .....	5
Definicje wyzwalaczy, procedur, funkcji .....	6
Wyzwalacze .....	6
Procedury .....	6
Funkcje .....	8
Skrypty testujące bazę danych .....	8
Wyzwalacze .....	9
Procedury .....	10
Funkcje .....	11
Analiza rozwiązania.....	12
Kod źródłowy w języku Java .....	13
Podsumowanie .....	13

## Krótki opis rozwiązania

Celem wykonywanego projektu jest stworzenie hotelowej bazy danych, która będzie odpowiedzialna za rezerwacje wykonywane w hotelu.

Założyliśmy, że rezerwację może dokonać jedna osoba (organizator), której „podlegają” inne osoby i razem tworzą grupę. Jedna grupa może dokonać wielu rezerwacji, ponieważ ilość osób przyjeżdżających może być większa niż maksymalna pojemność pokoi w hotelu. Podczas dokonywania rezerwacji można skorzystać ze zniżki sezonowej (założyliśmy, że zniżki są sezonowe zmieniają się co roku – w naszym przypadku są to zniżki na rok 2021) oraz z atrakcji, które oferuje hotel. Rezerwację mogą przyjąć kierownicy odpowiedniej zmiany oraz recepcjoniści. Podczas rezerwacji dokonuje się wyboru pokoju, ze względu na typ oraz status. Wysokość kosztu pobytu zależy od wyboru typu pokoju. Status pokoju opisuje, czy pokoje są wolne, zajęte lub remoncie. Każdy pokój znajduje się na odpowiednim piętrze i jest to odnotowane w bazie danych. W hotelu jest wielu pracowników, którzy pracują na przypisanych im piętrach, co ułatwia zarządzanie ich pracą, ponieważ hotel jest dość duży. Pracownicy zajmują odpowiednie stanowiska i zamieszkują pewne adresy.

W kolejnych rozdziałach zostały opisane poszczególne kroki, które umożliwiły nam stworzenie hotelowej bazy danych.

## Model ER, model relacyjny

W tym podpunkcie zajęliśmy się stworzeniem modelu ER naszego projektu, korzystając z odpowiednich narzędzi. Jak widać na Rys. 1. zdefiniowaliśmy 12 różnych encji.

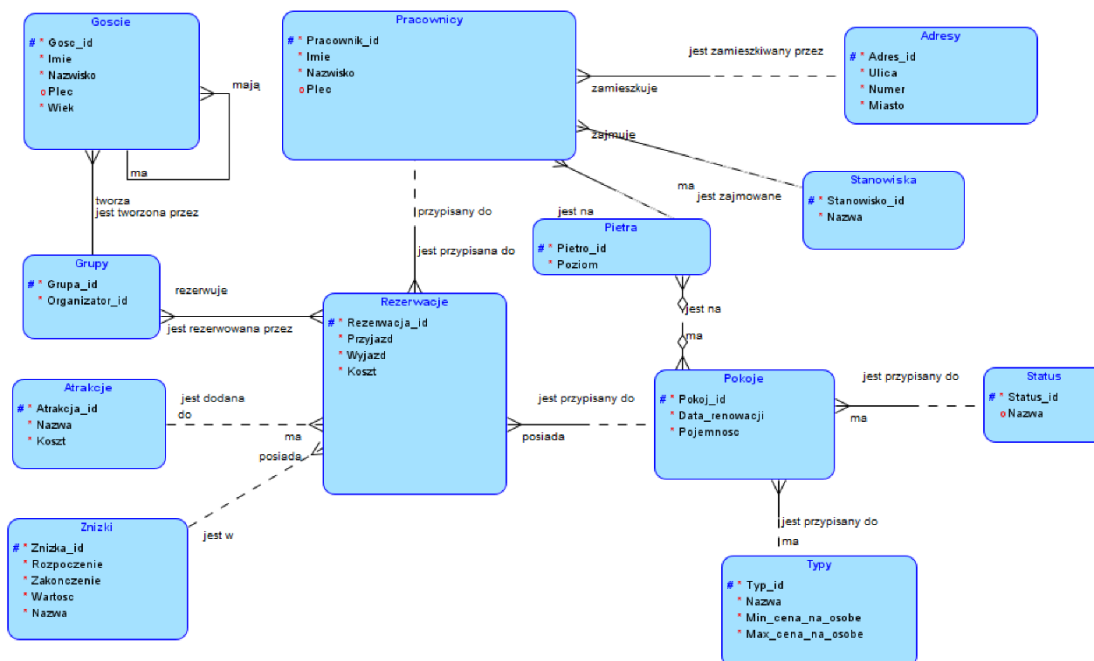
- *Goscie* – encja, która posiada informacje o gościach hotelu
- *Grupy* – encja, która posiada informacje o grupie, która będzie przebywać w hotelu.
- *Atrakcje* – encja, która posiada informację o atrakcjach oferowanych przez hotel
- *Znizki* – encja, która posiada informację o zniżkach oferowanych przez hotel (są to zniżki sezonowe).
- *Rezerwacje* – encja, która posiada informację o rezerwacjach w hotelu
- *Pracownicy* – encja, która posiada informację o pracownikach hotelu
- *Adresy* – encja, która posiada informację o adresach zamieszkania pracowników
- *Stanowiska* – encja, która posiada informację o zajmowanych stanowiskach w hotelu
- *Pietra* – encja, która posiada informację o piętrach w hotelu
- *Pokoje* – encja, która opisuje pokoje w hotelu
- *Status* – encja, która posiada informację o statusach pokoi hotelowych
- *Typy* – encja, która posiada informację o typach pokoi hotelowych

Wyżej wymienione encje zostały połączone między sobą pewnymi relacjami, które również można odczytać z Rys. 1.

### ❖ Relacja 1:N

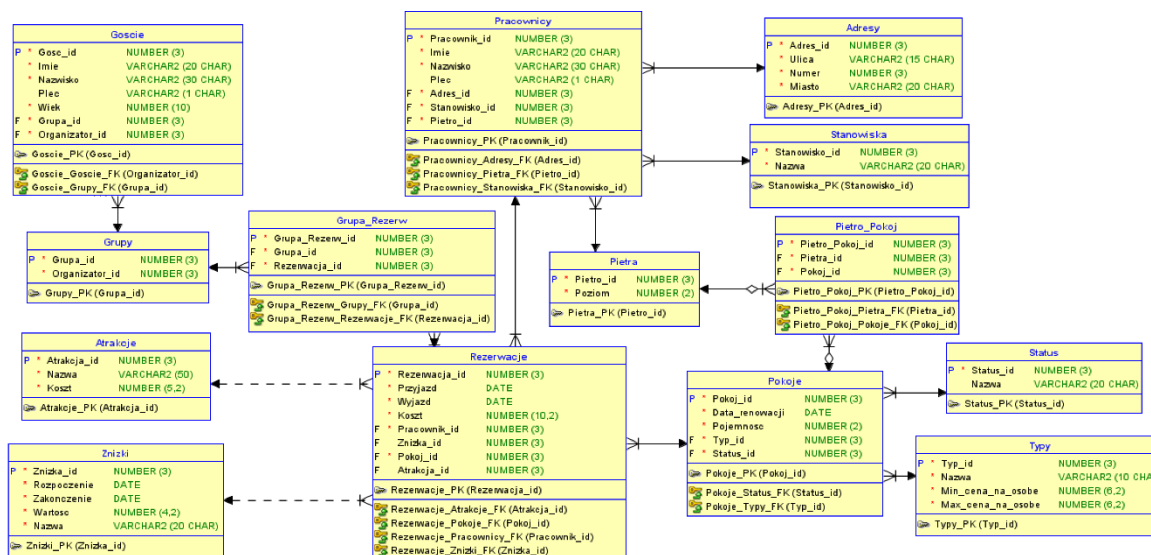
- Goście tworzą grupę. Grupa jest tworzona przez Gości
- Goście mają organizatora. Organizator ma Grupę.
- Atrakcja może być dodana do Rezerwacji. Rezerwacja może mieć Atrakcję.
- Zniżka może być dodana do Rezerwacji. Rezerwacja może mieć Zniżkę.
- Rezerwacja jest przypisana do Pracownika. Pracownik może być przypisany do Rezerwacji.

- Rezerwacja posiada Pokoj. Pokoj może być przypisany do Rezerwacji.
  - Pracownik zamieszkuje pewien Adres. Adres może być zamieszkiwany przez Pracownika.
  - Pracownik zajmuje pewne Stanowisko. Stanowisko może być zajmowane przez Pracownika.
  - Pracownik jest przypisany do Pietra, Pietro może mieć przypisanego Pracownika.
  - Pokoj jest przypisany do Typu. Typ może być przypisany do Pokoju.
  - Pokoj ma pewien Status. Status może być przypisany do Pokoju.
- ❖ Relacja N:M
- Grupa ma Rezerwację. Rezerwacja jest dokonywana przez Grupę.
  - Pokoj jest na Pietrze. Pietro ma Pokoje.



Rys. 1. Model ER wraz z relacjami

Kolejnym krokiem było stworzenie modelu relacyjnego na podstawie modelu ER. Dzięki opcji *Engineer to Relation Model* (🔗) wygenerowany został model relacyjny jak na Rys. 2. Jak łatwo zauważyć stworzyły się dodatkowo dwie tabele (*Grupa\_Rezerw* i *Pietro\_Pokoje*). Te tabele odpowiadają za relację N:M, które zostały opisane wyżej.



Rys. 2. Model relacyjny

Oba schematy zostały załączone do załącznika *Krakowiak\_Rancew.zip*

Skrypty do założenia schematu bazy danych

Kolejnym zadaniem w projekcie było stworzenie schematu bazy danych (skrypt DDL). W tym celu wykorzystano również gotowe funkcje/opcje do generacji takiego skryptu (*File -> Data Modeler -> Export-> DDL File*). Dzięki temu otrzymaliśmy prawie gotowy skrypt. Do tego skryptu dodałyśmy procedurę, która przygotowuje środowisko do użytku przy prezentacji projektu tzn. usuwa tabele, jeśli takie istniały wcześniej.

Opisany skrypt został załączony do załącznika *Krakowiak\_Rancew.zip* jako plik *skrypt\_DDL*.

Skrypty do załadowania danych

Następnie należało stworzyć skrypty do załadowania danych. Przed przystąpieniem tego zadania należało przemyśleć kolejność wykonywania poleceń, aby nie naruszać więzów integralności. W tym etapie głównie korzystano z polecenia *INSERT*, która powodowała dodanie kolejnych krotek do tabel oraz *COMMIT*, która potwierdzała wszystkie dodania.

Stworzone skrypty zostały załączone do załącznika *Krakowiak\_Rancew.zip* jako plik *skrypty\_DML*.

## Definicje wyzwalaczy, procedur, funkcji

Wszystkie stworzone niżej opisane definicje zostały załączone do załącznika *Krakowiak\_Rancew.zip* jako plik *definicje*.

### Wyzwalacze

W naszym projekcie zdefiniowano dwa wyzwalacze *ostrzezenie\_zly\_status* oraz *ostrzezenie\_dziecko*. Odpowiednio pierwsza odpowiada za powstrzymanie dodania nowej rezerwacji na pokój o statusie remont/wyłączony/zajęty (Rys. 3.). Natomiast drugi wyzwalacz ostrzega użytkownika, że dodany gość to dziecko i nie należy brać go pod uwagę (Rys. 4.).

```
101 CREATE OR REPLACE TRIGGER ostrzezenie_zly_status
102 BEFORE INSERT ON rezerwacje FOR EACH ROW
103 DECLARE
104     v_status_id pokoje.status_id%TYPE;
105 BEGIN
106     SELECT status_id
107     INTO v_status_id
108     FROM pokoje WHERE pokoj_id =: new.pokoj_id;
109
110     IF v_status_id = 901 OR v_status_id = 904 THEN
111         dbms_output.put_line('Pokój w remoncie/wyłączony');
112         raise_application_error(-20001, 'Rezerwacja niedokonana');
113     ELSIF v_status_id = 902 THEN
114         dbms_output.put_line('Pokój zajęty! Sprawdź do kiedy!');
115         raise_application_error(-20002, 'Rezerwacja niedokonana');
116     ELSE
117         dbms_output.put_line('Rezerwacja poszła pomyślnie!');
118     END IF;
119 END;
120 /
```

Rys. 3. Wyzwalacz *ostrzezenie\_zly\_status*

```
121 CREATE OR REPLACE TRIGGER ostrzezenie_dziecko
122 BEFORE INSERT ON goscie FOR EACH ROW
123 DECLARE
124 BEGIN
125     IF :new.wiek < 5 THEN
126         dbms_output.put_line('Dziecko! Nie liczone w koszty!');
127     END IF;
128 END;
129 /
```

Rys. 4. Wyzwalacz *ostrzezenie\_ilosc\_osob*

### Procedury

Następnie stworzyliśmy procedury. Pierwsza z nich *zmiana\_statusu\_remont* (*pok\_id* *NUMBER*) odpowiada za zmianę statusu pokoju na 'remont,' jeśli jest on 'wolny' i jego ostatni

remont był wykonany przed 2015 (Rys. 5). Natomiast druga (Rys. 6) odpowiada za zmianę atrakcji hotelowej wraz z update kosztów pobytu.

```
5 CREATE OR replace PROCEDURE zmiana_statusu_remont(pok_id NUMBER)
6 AS
7     v_status_id pokoje.status_id%TYPE;
8     v_data_renowacji pokoje.data_renowacji%TYPE;
9 BEGIN
10     SELECT status_id, v_data_renowacji
11     INTO v_status_id, v_data_renowacji
12     FROM pokoje
13     WHERE pokoj_id = pok_id;
14
15     IF v_status_id = 903 AND EXTRACT(YEAR FROM v_data_renowacji) < 2015 THEN
16         UPDATE pokoje SET status_id = 901;
17     ELSE
18         dbms_output.put_line ('Pokój nie może być oddany do remontu');
19     END IF;
20 END;
21 /
```

Rys. 5. Procedura zmiana\_statusu\_remont

```
25 CREATE OR replace PROCEDURE zmiana_atrakcji(rezerw_id NUMBER, atrak_id NUMBER)
26 AS
27     v_atrak_id_old rezerwacje.atrakcja_id%TYPE;
28     v_atrak_koszt_old atrakcje.koszt%TYPE;
29     v_atrak_koszt_new atrakcje.koszt%TYPE;
30     v_roznica atrakcje.koszt%TYPE;
31 BEGIN
32     SELECT a.atrakcja_id, a.koszt
33     INTO v_atrak_id_old, v_atrak_koszt_old
34     FROM rezerwacje r JOIN atrakcje a ON r.atrakcja_id = a.atrakcja_id
35     WHERE r.rezerwacja_id = rezerw_id;
36
37     SELECT koszt
38     INTO v_atrak_koszt_new
39     FROM atrakcje
40     WHERE atrakcja_id = atrak_id;
41
42     v_roznica := v_atrak_koszt_new - v_atrak_koszt_old;
43
44     IF v_atrak_id_old != atrak_id THEN
45         UPDATE rezerwacje SET atrakcja_id = atrak_id, koszt = koszt + v_roznica
46         WHERE rezerwacja_id = rezerw_id;
47         dbms_output.put_line ('Uaktualniono atrakcję');
48     ELSE
49         dbms_output.put_line ('Atrakcja została już wybrana!');
50     END IF;
51 END;
52 /
```

Rys. 6. Procedura zmiana\_atrakcji

## Funkcje

Kolejnym krokiem było stworzenie funkcji. Nasze funkcje odpowiadają za tworzenie kodów dostępu dla każdego pracownika do systemu komputerowego w hotelu (Rys. 7) oraz za wyznaczaniu ile dany pracownik zarejestrował rezerwacji (Rys. 8).

```
57 CREATE OR REPLACE FUNCTION kod_dostepu(prac_id NUMBER)
58 RETURN VARCHAR2
59 AS
60     v_kod_p VARCHAR(5);
61     v_kod VARCHAR(5);
62 BEGIN
63     SELECT CONCAT(SUBSTR(imie,1,1), SUBSTR(nazwisko,1,1))
64     INTO v_kod_p
65     FROM pracownicy
66     WHERE pracownik_id = prac_id;
67
68     SELECT CONCAT(TO_CHAR(pracownik_id), v_kod_p)
69     INTO v_kod
70     FROM pracownicy
71     WHERE pracownik_id = prac_id;
72
73 RETURN v_kod;
74 END;
75 /
```

Rys. 7. Funkcja kod\_dostepu

```
78 CREATE OR REPLACE FUNCTION ilosc_rezer(prac_id NUMBER)
79 RETURN NUMBER
80 AS
81     c_ilosc NUMBER;
82 BEGIN
83     SELECT COUNT(rezerwacja_id)
84     INTO c_ilosc
85     FROM rezerwacje
86     WHERE pracownik_id = prac_id;
87
88     IF c_ilosc > 1 THEN
89         dbms_output.put_line('Pracownik o id: ' || prac_id || ' dokonał ' || c_ilosc || ' rezerwacji. ');
90     ELSE
91         dbms_output.put_line('Pracownik nie ma uprawnień do dokonywania rezerwacji!');
92     END IF;
93 RETURN c_ilosc;
94 END;
95 /
```

Rys. 8. Funkcja ilosc\_rezer

## Skrypty testujące bazę danych

Jednym z końcowych etapów jest sprawdzenie działania naszej hotelowej bazy danych oraz zdefiniowanych wyzwalaczy, procedur, funkcji. W tym rozdziale zajmiemy się sprawdzeniem działania wyzwalaczy, procedur, funkcji (testy znajdują się w pliku *definicje*). Natomiast sprawdzenie różnych podzapytań, wywoływanie kursorów znajduje się w załączniku *Krakowiak\_Rancew.zip* jako plik *testy*.



Wyzwalacze

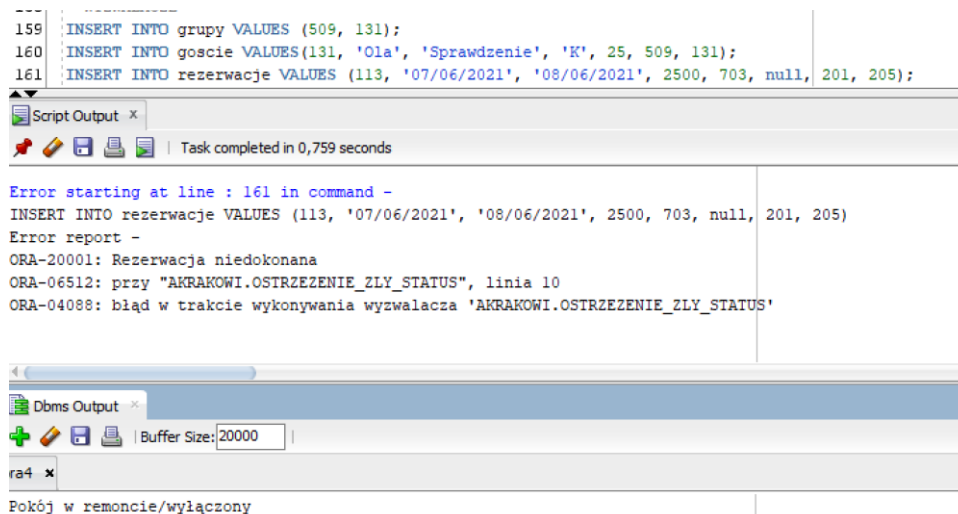
Zadanie:

*Stwórz grupę, która będzie posiadała jedną osobę oraz złóż rezerwację na pokój 201*

Polecenie:

```
INSERT INTO grupy VALUES (509, 131);
INSERT INTO goscie VALUES(131, 'Ola', 'Sprawdzenie', 'K', 25, 509, 131);
INSERT INTO rezerwacje VALUES (113, '07/06/2021', '08/06/2021', 2500, 703, null, 201, 205);
```

Wynik:



Rys. 9. Wynik po próbie dodania rezerwacji - działanie wyzwalacza `ostrzezenie_zly_status`

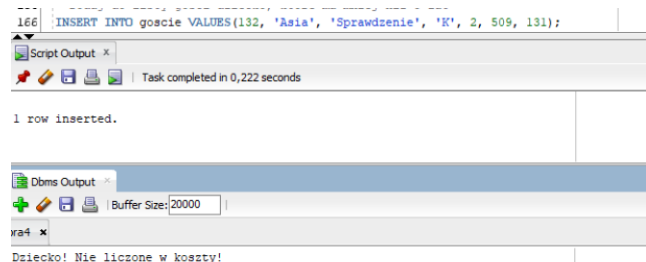
Zadanie:

*Dodaj do listy gości dziecko, które ma mniej niż 5 lat*

Polecenie:

```
INSERT INTO goscie VALUES(132, 'Asia', 'Sprawdzenie', 'K', 2, 509, 131);
```

Wynik:



Rys. 10. Wynik po zadziałaniu wyzwalacza `ostrzezenie_dziecko`

## Procedury

### Zadanie:

Wywołaj procedurę, która umożliwi zmianę atrakcji w rezerwacji 104 na atrakcję SPA

### Polecenie:

```
BEGIN
    zmiana_atrakcji(104,202);
END;
/
```

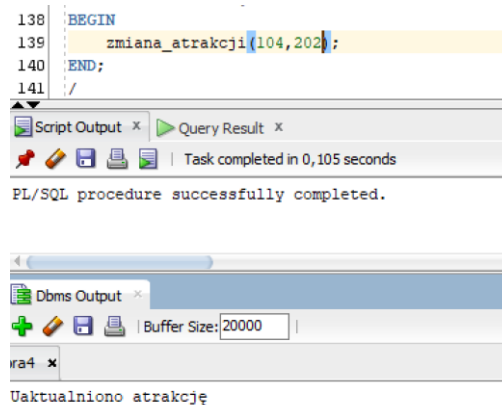
### Wynik:

przed

4	104	01/07/2021	14/07/2021	8000	705	208	205	201
---	-----	------------	------------	------	-----	-----	-----	-----

Rys. 11. Rezerwacja 104 przed aktualizacją

po



Rys. 12. Wynik po wywołaniu procedury zmiana\_atrakcji

4	104	01/07/2021	14/07/2021	8230	705	208	205	202
---	-----	------------	------------	------	-----	-----	-----	-----

Rys. 13. Rezerwacja 104 po aktualizacji

### Zadanie:

Wywołaj procedurę, która umożliwi zmienienie statusu pokoju o id 232 na Remont

### Polecenie:

```
BEGIN
    zmiana_statusu_remont(232);
END;
/
```

Wynik:

```
144 | BEGIN
145 |     zmiana_statusu_remont(232);
146 | END;
147 | /
```

Script Output x Query Result x

Task completed in 0,098 seconds

PL/SQL procedure successfully completed.

Dbms Output x

Buffer Size: 20000

ora4 x

Uaktualniono atrakcję

Pokój nie może być oddany do remontu

Rys. 14. Wynik po wywołaniu procedury `zmiana_statusu_remont`

## Funkcje

Zadanie:

*Pokaż kody dostępu dla pracowników, którzy pracują na stanowisku Kucharz*

Polecenie:

```
SELECT kod_dostepu(p.pracownik_id) AS kod
FROM pracownicy p JOIN stanowiska s USING (stanowisko_id)
WHERE s.nazwa = 'Kucharz';
```

Wynik:

KOD
706SW
707GB
708KC
709ZS
710KS
711LS

6 rows selected.

Rys. 15. Wynik po wywołaniu funkcji `kod_dostepu`

Zadanie:

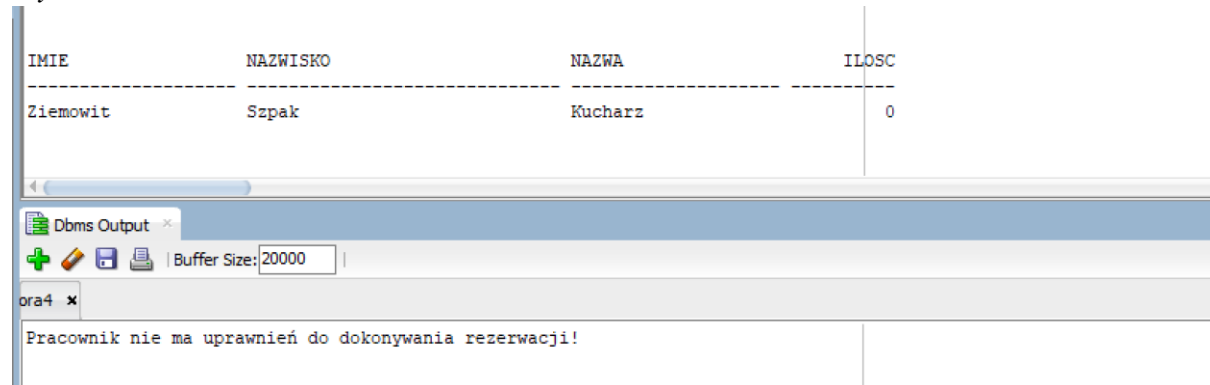
*Pokaż imię, nazwisko, nazwę stanowiska oraz ilość dokonany rezerwacji przez pracownika o danym ID*

*Polecenie:*

```
SELECT p.imie, p.nazwisko, s.nazwa ,ilosc_rezer(pracownik_id) AS ilosc
FROM pracownicy p JOIN stanowiska s USING (stanowisko_id)
WHERE pracownik_id = 709;
```

*Wynik:*

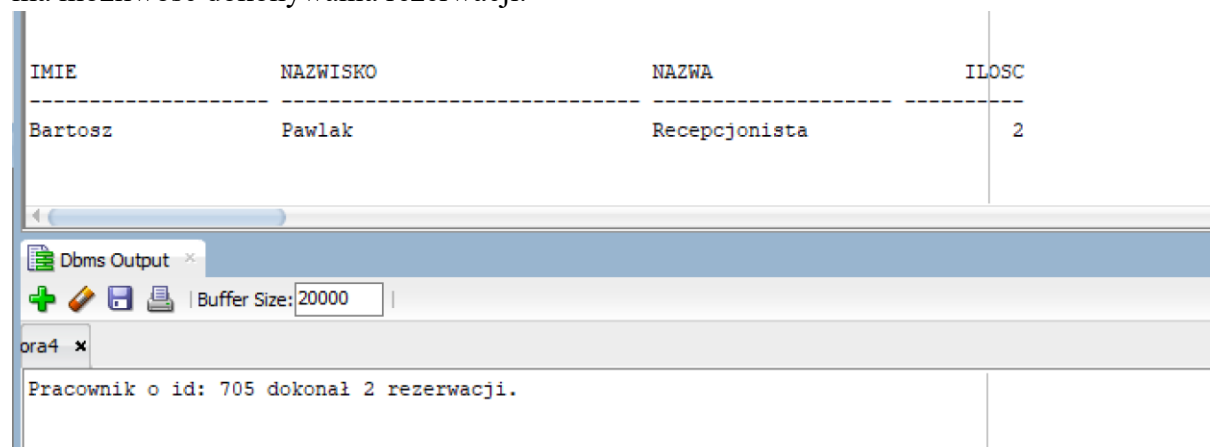
IMIE	NAZWISKO	NAZWA	ILOSC
Ziemowit	Szpak	Kucharz	0



*Rys. 16. Wynik po wywołaniu funkcji ilosc\_rezer dla pracownik\_id = 709*

Do sprawdzenia poprawności działania funkcji wykonaliśmy zapytanie dla pracownika, który ma możliwość dokonywania rezerwacji.

IMIE	NAZWISKO	NAZWA	ILOSC
Bartosz	Pawlak	Recepcjonista	2



*Rys. 17. Wynik po wywołaniu funkcji ilosc\_rezer dla pracownik\_id = 705*

## Analiza rozwiązania

W naszym projekcie jest pokazana niewielka ilość możliwości jaką daje nam ten temat. Naszą bazę można rozbudować o nowe tabele lub przebudować na mniejszą ilość tabel lub inaczej podejść do tego tematu. Jest to dość szeroki temat. Mamy wrażenie, że powinniśmy inaczej podejść do tego tematu, dzięki czemu łatwiej byłoby nam formułowanie nowych funkcji, wyzwalaczy itp.

## Kod źródłowy w języku Java

Nasz kod źródłowy umieściliśmy na GitLab oraz w załączniku *Krakowiak\_Rancew.zip*. W naszej aplikacji występuje kilka metod:

- `setConnection()` – umożliwia nawiązuje połączenie z bazą danych
- `closeConnection()` – metoda do zamknięcia połączenia z bazą danych
- `pokazGosci()` – metoda umożliwiająca wypis wszystkich gości w hotelu, którzy znajdują się w tabeli *Goscie*
- `pokazGrupeGosci()` – metoda, która umożliwia wypis gości należących do konkretnej grupy (użytkownik podaje numer grupy, którą chce zobaczyć)
- `aktualizacjaCenAtrakcji()` – metoda, która umożliwia aktualizację cen atrakcji hotelowych
- `kodDostep()` – metoda, która korzysta z funkcji *kod\_dostepu*
- `main(String[] args)` – główna metoda aplikacji, w której możemy wywołać poszczególne metody

Aplikacja polega na połączeniu się z bazą danych i wykonaniu pewnych metod (w kodzie odpowiednią metodę odkomentujemy, aby zobaczyć jej działanie) oraz na poprawnym rozłączaniu się z bazą.

Ważne! Metody `setConnection()` i `closeConnection()` zawsze powinny brać udział w uruchamianiu aplikacji.

Dodatkowo wstępuje plik `infoConn.properties`, w którym należy podać własne *username* oraz *password* do SQLDevelopera, który umożliwi połączenie się z bazą.

### WAŻNE!

Przed uruchomieniem aplikacji należy w SQLDeveloper otworzyć i uruchomić pliki: *skrypt\_DDL*, *skrypty\_DML* oraz *definicje*.

### Podsumowanie

#### Załącznik *Krakowiak\_Rancew.zip*

schemat modelu ER i modelu relacyjnego jako *projekt\_Krakowiak\_Rancew*

skrypt tworzący odpowiednie tabele – *skrypt\_DDL*

skrypt uzupełniający tabele danymi - *skrypty\_DML*

definicje procedur, wyzwalaczy, funkcji oraz ich sprawdzenie – *definicje*

pozostałe sprawdzenie – *testy*

główna klasa aplikacji *HotelApp* oraz `infoConn.properties`

#### GitLab

cała aplikacja *HotelApp*