

## Oracle PL/SQL,- ćwiczenie

### PL/SQL – programowanie proceduralne, widoki, procedury, triggerzy

#### 1. Tabele

Wycieczki (id\_wycieczki, nazwa, kraj, data, opis, liczba\_miejsc)

Osoby(id\_osoby, imie, nazwisko, pesel, kontakt)

Rezerwacje(nr\_rezerwacji, id\_wycieczki, id\_osoby, status)

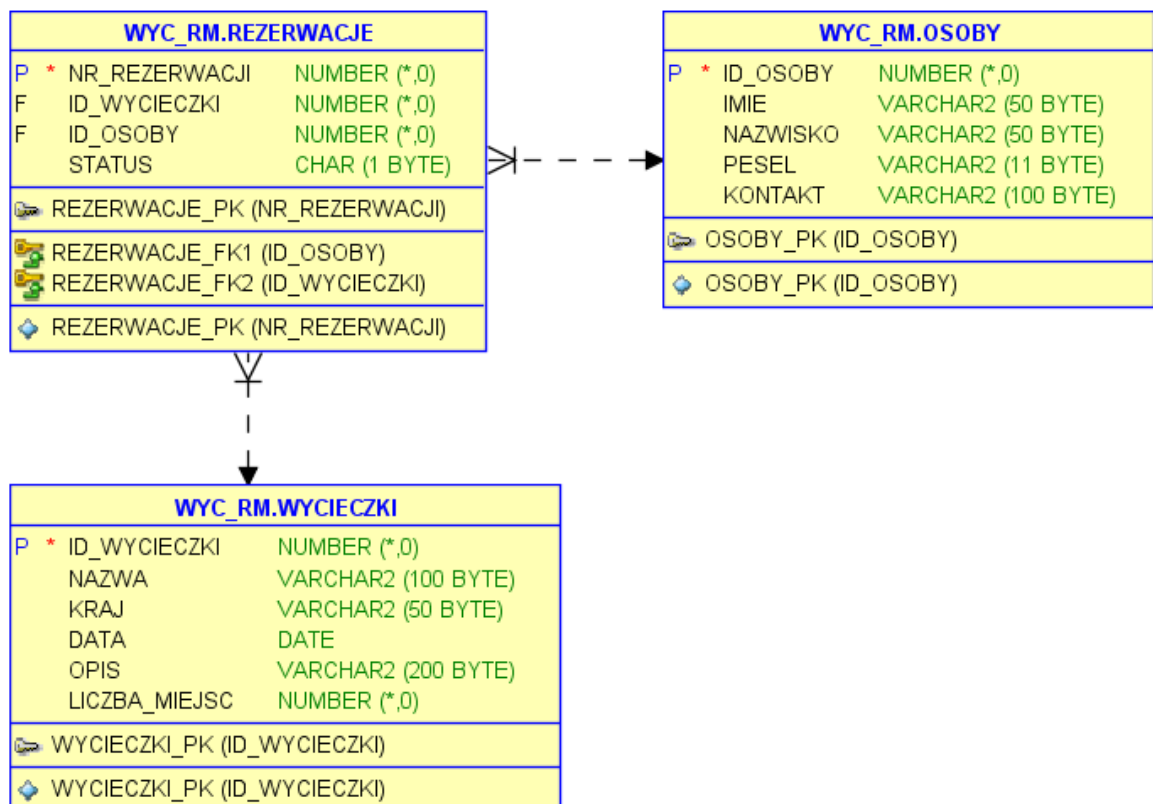
Pole status w tabeli Rezerwacje może przyjmować jedną z 4 wartości

N – Nowa

P – Potwierdzona

Z – Potwierdzona i zapłacona

A – Anulowana



```

-- USER SQL
CREATE USER wyc IDENTIFIED BY "oracle"
DEFAULT TABLESPACE "USERS"
TEMPORARY TABLESPACE "TEMP";

-- ROLES
GRANT "DBA" TO wyc WITH ADMIN OPTION;
ALTER USER wyc DEFAULT ROLE "DBA";

--TABLES
CREATE TABLE OSOBY
(
    ID_OSOBY INT GENERATED ALWAYS AS IDENTITY NOT NULL
, IMIE VARCHAR2(50)
, NAZWISKO VARCHAR2(50)
, PESEL VARCHAR2(11)
, KONTAKT VARCHAR2(100)
, CONSTRAINT OSOBY_PK PRIMARY KEY
(
    ID_OSOBY
)
ENABLE
);

CREATE TABLE WYCIECZKI
(
    ID_WYCIECZKI INT GENERATED ALWAYS AS IDENTITY NOT NULL
, NAZWA VARCHAR2(100)
, KRAJ VARCHAR2(50)
, DATA DATE
, OPIS VARCHAR2(200)
, LICZBA_MIEJSC INT
, CONSTRAINT WYCIECZKI_PK PRIMARY KEY
(
    ID_WYCIECZKI
)
ENABLE
);

CREATE TABLE REZERWACJE
(
    NR_REZERWACJI INT GENERATED ALWAYS AS IDENTITY NOT NULL
, ID_WYCIECZKI INT
, ID_OSOBY INT
, STATUS CHAR(1)
, CONSTRAINT REZERWACJE_PK PRIMARY KEY
(
    NR_REZERWACJI
)
ENABLE
);

```

```

ALTER TABLE REZERWACJE
ADD CONSTRAINT REZERWACJE_FK1 FOREIGN KEY
(
    ID_OSOBY
)
REFERENCES OSOBY
(
    ID_OSOBY
)
ENABLE;

```

```

ALTER TABLE REZERWACJE
ADD CONSTRAINT REZERWACJE_FK2 FOREIGN KEY
(
    ID_WYCIECZKI
)
REFERENCES WYCIECZKI
(
    ID_WYCIECZKI
)
ENABLE;

```

```

ALTER TABLE REZERWACJE
ADD CONSTRAINT REZERWACJE_CHK1 CHECK
(status IN ('N','P','Z','A'))
ENABLE;

```

## 2. Wypełniamy tabele przykładowymi danymi danymi

4 wycieczki

10 osób

10 rezerwacji

```

INSERT INTO osoby (imie, nazwisko, pesel, kontakt)
VALUES ('Adam', 'Kowalski', '87654321', 'tel: 6623');

```

```

INSERT INTO osoby (imie, nazwisko, pesel, kontakt)
VALUES ('Jan', 'Nowak', '12345678', 'tel: 2312, dzwonić po 18.00');

```

```

INSERT INTO wycieczki (nazwa, kraj, data, opis, liczba_miejsc)
VALUES ('Wycieczka do Paryza', 'Francja', '2016-01-01', 'Ciekawa wycieczka ...', 3);

```

```

INSERT INTO wycieczki (nazwa, kraj, data, opis, liczba_miejsc)
VALUES ('Piękny Kraków', 'Polska', '2017-02-03', 'Najciekawa wycieczka ...', 2);

```

```

INSERT INTO wycieczki (nazwa, kraj, data, opis, liczba_miejsc)
VALUES ('Wieliczka', 'Polska', '2017-03-03', 'Zadziwiająca kopalnia ...', 2);

```

--UWAGA

--W razie problemów z formatem daty można użyć funkcji TO\_DATE

```

INSERT INTO wycieczki (nazwa, kraj, data, opis, liczba_miejsc)
VALUES ('Wieliczka2', 'Polska', TO_DATE('2017-03-03', 'YYYY-MM-DD'),
    'Zadziwiająca kopalnia ...', 2);

```

```
INSERT INTO rezerwacje(id_wycieczki, id_osoby, status)
VALUES (1,1,'N');
```

```
INSERT INTO rezerwacje(id_wycieczki, id_osoby, status)
VALUES (2,2,'P');
```

3. Tworzenie widoków. Należy przygotować kilka widoków ułatwiających dostęp do danych

- a) wycieczki\_osoby(kraj,data, nazwa\_wycieczki, imie, nazwisko,status\_rezerwacji)
- b) wycieczki\_osoby\_potwierdzone (kraj,data, nazwa\_wycieczki, imie, nazwisko,status\_rezerwacji)
- c) wycieczki\_przyszle (kraj,data, nazwa\_wycieczki, imie, nazwisko,status\_rezerwacji)
- d) wycieczki\_miejsca(kraj,data, nazwa\_wycieczki,liczba\_miejsc, liczba\_wolnych\_miejsc)
- e) dostępne\_wycieczki(kraj,data, nazwa\_wycieczki,liczba\_miejsc, liczba\_wolnych\_miejsc)
- f) rezerwacje\_do\_anulowania – lista niepotwierdzonych rezerwacji które powinny zostać anulowane, rezerwacje przygotowywane są do anulowania na tydzień przed wyjazdem)

```
CREATE VIEW wycieczki_osoby
AS
SELECT
    w.ID_WYCIECZKI,
    w.NAZWA,
    w.KRAJ,
    w.DATA,
    o.IMIE,
    o.NAZWISKO,
    r.STATUS
FROM WYCIECZKI w
JOIN REZERWACJE r ON w.ID_WYCIECZKI = r.ID_WYCIECZKI
JOIN OSOBY o ON r.ID_OSOBY = o.ID_OSOBY;
```

Proponowany zestaw widoków można rozbudować wedle uznania/potrzeb

4. Tworzenie procedur/funkcji pobierających dane. Podobnie jak w poprzednim przykładzie należy przygotować kilka procedur ułatwiających dostęp do danych

- a) uczestnicy\_wycieczki(id\_wycieczki), procedura ma zwracać podobny zestaw danych jak widok wycieczki\_osoby
- b) rezerwacje\_osoby(id\_osoby), procedura ma zwracać podobny zestaw danych jak widok wycieczki\_osoby
- c) przyszle\_rezerwacje\_osoby(id\_osoby)
- d) dostępne\_wycieczki(kraj, data\_od, data\_do)

Procedury/funkcje powinny zwracać tabelę/zbiór wynikowy

Należy zwrócić uwagę na kontrolę parametrów (np. jeśli parametrem jest id\_wycieczki to należy sprawdzić czy taka wycieczka istnieje)

5. Tworzenie procedur modyfikujących dane. Należy przygotować zestaw procedur pozwalających na modyfikację danych oraz kontrolę poprawności ich wprowadzania

- a) `dodaj_rezerwacje(id_wycieczki, id_osoby)`, procedura powinna kontrolować czy wycieczka jeszcze się nie odbyła, i czy są wolne miejsca
- b) `zmien_status_rezerwacji(id_rezerwacji, status)`, procedura kontrolować czy możliwa jest zmiana statusu, np. zmiana statusu już anulowanej wycieczki (przywrócenie do stanu aktywnego nie zawsze jest możliwe)
- c) `zmien_liczbe_miejsc(id_wycieczki, liczba_miejsc)`, nie wszystkie zmiany liczby miejsc są dozwolone, nie można zmniejszyć liczby miejsc na wartość poniżej liczby zarezerwowanych miejsc

Należy rozważyć użycie transakcji

Należy zwrócić uwagę na kontrolę parametrów (np. jeśli parametrem jest `id_wycieczki` to należy sprawdzić czy taka wycieczka istnieje, jeśli robimy rezerwację to należy sprawdzać czy są wolne miejsca)

6. Dodajemy tabelę dziennikującą zmiany statusu rezerwacji

`rezerwacje_log(id, id_rezerwacji, data, status)`

Należy zmienić warstwę procedur modyfikujących dane tak aby dopisywały informację do dziennika

7. Zmiana struktury bazy danych, w tabeli wycieczki dodajemy redundantne pole `liczba_wolnych_miejsc`

Należy zmodyfikować zestaw widoków. Proponuję dodać kolejne widoki (np. z sufiksem 2), które pobierają informację o wolnych miejscach z nowo dodanego pola.

Należy napisać procedurę `przelicz` która zaktualizuje wartość liczby wolnych miejsc dla już istniejących danych

Należy zmodyfikować warstwę procedur pobierających dane, podobnie jak w przypadku widoków.

Należy zmodyfikować procedury wprowadzające dane tak aby korzystały/aktualizowały pole `liczba_wolnych_miejsc` w tabeli wycieczki

Najlepiej to zrobić tworząc nowe wersje (np. z sufiksem 2)

8. Zmiana strategii zapisywania do dziennika rezerwacji. Realizacja przy pomocy triggerów

Należy wprowadzić zmianę która spowoduje że zapis do dziennika rezerwacji będzie realizowany przy pomocy triggerów

trigger obsługujący dodanie rezerwacji  
trigger obsługujący zmianę statusu  
trigger zabraniający usunięcia rezerwacji

Oczywiście po wprowadzeniu tej zmiany należy uaktualnić procedury modyfikujące dane.  
Najlepiej to zrobić tworząc nowe wersje (np. z sufiksem 3)

9. Zmiana strategii obsługi redundantnego pola liczba\_wolnych\_miejsc . realizacja przy pomocy triggerów

trigger obsługujący dodanie rezerwacji  
trigger obsługujący zmianę statusu  
trigger obsługujący zmianę liczby miejsc na poziomie wycieczki

Oczywiście po wprowadzeniu tej zmiany należy uaktualnić procedury modyfikujące dane.  
Najlepiej to zrobić tworząc nowe wersje (np. z sufiksem 3)

## Podsumowanie

Należy przygotować raport z wykonania ćwiczenia. Raport powinien zawierać polecenia SQL (między innymi kod widoków, procedur), wynik działania oraz krótki komentarz (jeśli jest potrzebny). Raport należy przesłać w formie pliku PDF