Oracle PL/SQL,- ćwiczenie

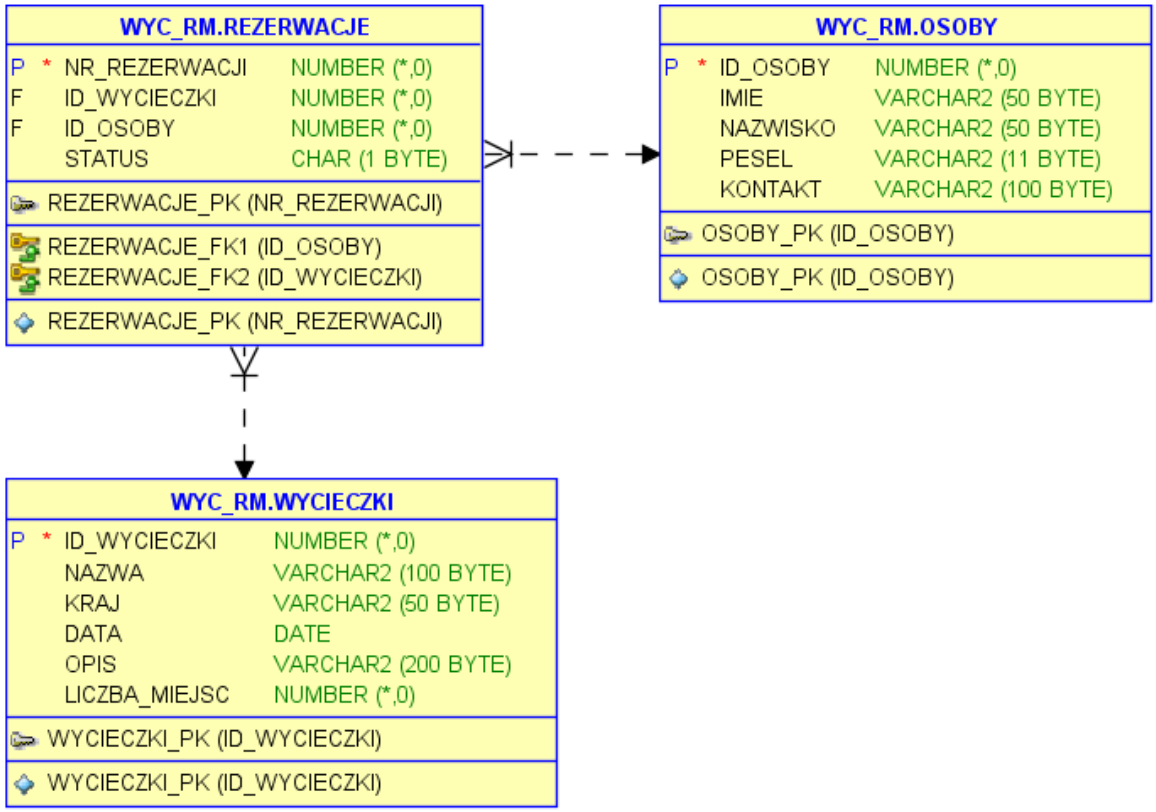
PL/SQL – programowanie proceduralne, widoki, procedury, triggery

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



Tworzenie użytkownika.
UWAGA: Te polecenia można wykonać jedynie w przypadku korzystania z własnego serwera Oracle

```
-- 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";
```

Tworzenie tabel.

```
--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. Wypełnianie tabele przykładowymi danymi danymi
   4 wycieczki
   10 osób
   10 rezerwacji

   Dane testowe powinny być różnorodne (wycieczki w przyszłości, wycieczki w przeszłości, rezerwacje o różnym statusie itp.) tak, żeby umożliwić testowanie napisanych procedur. W razie potrzeby należy zmodyfikować dane tak żeby przetestować różne przypadki.

```
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. Należy zwrócić uwagę na strukturę kodu (należy unikać powielania kodu)

   a) RzerwacjeWszystkie(kraj,data, nazwa_wycieczki, imie, nazwisko,status_rezerwacji)
   b) RezerwacjePotwierdzone (kraj,data, nazwa_wycieczki, imie, nazwisko,status_rezerwacji)
   c) RezerwacjeWPrzyszlosci (kraj,data, nazwa_wycieczki, imie, nazwisko,status_rezerwacji)
   d) WycieczkiMiejsca(kraj,data, nazwa_wycieczki,liczba_miejsc, liczba_wolnych_miejsc)
   e) WycieczkiDostepne(kraj,data, nazwa_wycieczki,liczba_miejsc, liczba_wolnych_miejsc)

```
CREATE VIEW RezerwacjeWszystkie
  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) UczestnicyWycieczki (id_wycieczki), procedura ma zwracać podobny zestaw danych jak widok RezerwacjeWszystkie

b) RezerwacjeOsoby(id_osoby), procedura ma zwracać podobny zestaw danych jak widok wycieczki_osoby
c) DostepneWycieczki(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). Podobnie jak w przypadku widoków należy unikać powielania kodu.

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

   a) DodajRezerwacje(id_wycieczki, id_osoby), procedura powinna kontrolować czy wycieczka jeszcze się nie odbyła, i czy sa wolne miejsca
   b) ZmienStatusRezerwacji(nr_rezerwacji, status), procedura kontrolować czy możliwa jest zmiana statusu, np. zmiana statusu już anulowanej wycieczki (przywrócenie do stanu aktywnego nie zawsze jest możliwa – może już nie być miejsc)
   c) ZmienLiczbeMiejsc(id_wycieczki, liczba_miejsc), nie wszystkie zmiany liczby miejsc są dozwolone, nie można zmniejszyć liczby miesc 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 itp..)

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 trierów

   triger obsługujący dodanie rezerwacji
   triger obsługujący zmianę statusu
   triger 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 trigerów

   triger obsługujący dodanie rezerwacji
   triger obsługujący zmianę statusu
   triger 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