# Oracle PL/Sql

widoki, funkcje, procedury, triggery ćwiczenie

Imiona i nazwiska autorów : Jakub Worek, Przemysław Popowski, Jakub Białecki

## Tabele

- `Trip` - wycieczki
  - `trip_id` - identyfikator, klucz główny
  - `trip_name` - nazwa wycieczki
  - `country` - nazwa kraju
  - `trip_date` - data
  - `max_no_places` - maksymalna liczba miejsc na wycieczkę
- `Person` - osoby
  - `person_id` - identyfikator, klucz główny
  - `firstname` - imię
  - `lastname` - nazwisko
- `Reservation` - rezerwacje
  - `reservation_id` - identyfikator, klucz główny
  - `trip_id` - identyfikator wycieczki
  - `person_id` - identyfikator osoby
  - `status` - status rezerwacji
    - `N` – New - Nowa
    - `P` – Confirmed and Paid – Potwierdzona  i zapłacona
    - `C` – Canceled - Anulowana
- `Log` - dziennik zmian statusów rezerwacji
  - `log_id` - identyfikator, klucz główny
  - `reservation_id` - identyfikator rezerwacji
  - `log_date` - data zmiany
  - `status` - status

```
create sequence s_person_seq
   start with 1
   increment by 1;

create table person
(
  person_id int not null
      constraint pk_person
         primary key,
  firstname varchar(50),
  lastname varchar(50)
)

alter table person
    modify person_id int default s_person_seq.nextval;
```

```
create sequence s_trip_seq
   start with 1
   increment by 1;

create table trip
(
  trip_id int  not null
     constraint pk_trip
         primary key,
  trip_name varchar(100),
  country varchar(50),
  trip_date date,
  max_no_places int
);

alter table trip
    modify trip_id int default s_trip_seq.nextval;
```

```
create sequence s_reservation_seq
   start with 1
   increment by 1;

create table reservation
(
  reservation_id int not null
      constraint pk_reservation
         primary key,
  trip_id int,
  person_id int,
  status char(1)
);

alter table reservation
    modify reservation_id int default s_reservation_seq.nextval;


alter table reservation
add constraint reservation_fk1 foreign key
```

```
( person_id ) references person ( person_id );

alter table reservation
add constraint reservation_fk2 foreign key
( trip_id ) references trip ( trip_id );

alter table reservation
add constraint reservation_chk1 check
(status in ('N','P','C'));
```

```
create sequence s_log_seq
   start with 1
   increment by 1;


create table log
(
    log_id int not null
         constraint pk_log
         primary key,
    reservation_id int not null,
    log_date date not null,
    status char(1)
);

alter table log
    modify log_id int default s_log_seq.nextval;

alter table log
add constraint log_chk1 check
(status in ('N','P','C')) enable;

alter table log
add constraint log_fk1 foreign key
( reservation_id ) references reservation ( reservation_id );
```

## Dane

Należy wypełnić tabele przykładowymi 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.

```
-- trip
insert into trip(trip_name, country, trip_date, max_no_places)
values ('Wycieczka do Paryza', 'Francja', to_date('2023-09-12', 'YYYY-MM-DD'), 3);

insert into trip(trip_name, country, trip_date,  max_no_places)
values ('Piekny Krakow', 'Polska', to_date('2025-05-03','YYYY-MM-DD'), 2);

insert into trip(trip_name, country, trip_date,  max_no_places)
values ('Znow do Francji', 'Francja', to_date('2025-05-01','YYYY-MM-DD'), 2);

insert into trip(trip_name, country, trip_date,  max_no_places)
values ('Hel', 'Polska', to_date('2025-05-01','YYYY-MM-DD'),  2);

-- person
insert into person(firstname, lastname)
values ('Jan', 'Nowak');

insert into person(firstname, lastname)
values ('Jan', 'Kowalski');

insert into person(firstname, lastname)
values ('Jan', 'Nowakowski');

insert into person(firstname, lastname)
values  ('Novak', 'Nowak');

-- reservation
-- trip1
insert  into reservation(trip_id, person_id, status)
values (1, 1, 'P');

insert into reservation(trip_id, person_id, status)
values (1, 2, 'N');

-- trip 2
insert into reservation(trip_id, person_id, status)
values (2, 1, 'P');

insert into reservation(trip_id, person_id, status)
values (2, 4, 'C');

-- trip 3
insert into reservation(trip_id, person_id, status)
values (2, 4, 'P');
```

proszę pamiętać o zatwierdzeniu transakcji

## Zadanie 0 - modyfikacja danych, transakcje

Należy przeprowadzić kilka eksperymentów związanych ze wstawianiem, modyfikacją i usuwaniem danych oraz wykorzystaniem transakcji

Skomentuj działanie transakcji. Jak działa polecenie `commit`, `rollback`?. Co się dzieje w przypadku wystąpienia błędów podczas wykonywania transakcji? Porównaj sposób programowania operacji wykorzystujących transakcje w Oracle PL/SQL ze znanym ci systemem/językiem MS SqlServer T-SQL

pomocne mogą być materiały dostępne tu: https://upel.agh.edu.pl/mod/folder/view.php?id=214774 w szczególności dokument: `1_modyf.pdf`

```
Transakcje to mechanizm, który umożliwia grupowanie operacji na danych (takich jak wstawianie, modyfikacja, usuwanie) w jedną jednostkę, która jest wykonana albo w całości,
albo wcale, zapewniając tym samym spójność danych. Główne polecenia używane w zarządzaniu transakcjami to:

Commit - zatwierdza wszystkie zmiany dokonane w ramach transakcji, czyniąc je trwałymi i widocznymi dla innych użytkowników.
Rollback - cofa wszystkie zmiany dokonane w ramach transakcji, przywracając stan danych do momentu przed rozpoczęciem transakcji.
W przypadku wystąpienia błędów podczas transakcji żadna operacja nie zostaje zatwierdzona.
```

```
begin
    insert into person(firstname, lastname) values  ('Arnold', 'Schwartzenegger');
    insert into person(firstname, lastname) values ('Rocky', 'Balboa');
    rollback;
end;

-- Ta operacja mimo że poprawna nie zostanie zapisana w bazie z powodu rollbacka.

begin
    insert into person(firstname, lastname) values  ('Arnold', 'Schwartzenegger');
    insert into person(firstname, lastname) values ('Rocky', 'Balboa');
    commit;
end;
```

## Zadanie 1 - widoki

Tworzenie widoków. Należy przygotować kilka widoków ułatwiających dostęp do danych. Należy zwrócić uwagę na strukturę kodu (należy unikać powielania kodu)

Widoki:

- `vw_reservation`
  - widok łączy dane z tabel: `trip`, `person`, `reservation`
  - zwracane dane: `reservation_id`, `country`, `trip_date`, `trip_name`, `firstname`, `lastname`, `status`, `trip_id`, `person_id`
- `vw_trip`
  - widok pokazuje liczbę wolnych miejsc na każdą wycieczkę
  - zwracane dane: `trip_id`, `country`, `trip_date`, `trip_name`, `max_no_places`, `no_available_places` (liczba wolnych miejsc)
- `vw_available_trip`
  - podobnie jak w poprzednim punkcie, z tym że widok pokazuje jedynie dostępne wycieczki (takie które są w przyszłości i są na nie wolne miejsca)

Proponowany zestaw widoków można rozbudować wedle uznania/potrzeb

- np. można dodać nowe/pomocnicze widoki
- np. można zmienić def. widoków, dodając nowe/potrzebne pola

## Zadanie 1 - rozwiązanie

```
CREATE VIEW vw_reservation AS
SELECT
    reservation_id,
    country,
    trip_date,
    trip_name,
    firstname,
    lastname,
    status,
    RESERVATION.trip_id,
    RESERVATION.person_id
FROM RESERVATION
JOIN TRIP ON RESERVATION.trip_id = TRIP.trip_id
JOIN PERSON ON RESERVATION.person_id = PERSON.person_id;

CREATE VIEW vw_trip AS
SELECT
    t.trip_id,
    t.country,
    t.trip_date,
    t.trip_name,
    t.max_no_places,
    (t.max_no_places - COUNT(r.status)) AS no_available_places
FROM
    trip t
LEFT JOIN
    reservation r ON t.trip_id = r.trip_id AND r.status IN ('N', 'P')
GROUP BY
    t.trip_id, t.country, t.trip_date, t.trip_name, t.max_no_places;

CREATE VIEW vw_available_trip AS
SELECT
    t.trip_id,
    t.country,
    t.trip_date,
    t.trip_name,
    t.max_no_places,
    (t.max_no_places - COUNT(r.status)) AS no_available_places
FROM
    trip t
LEFT JOIN
    reservation r ON t.trip_id = r.trip_id AND r.status IN ('N', 'P')
WHERE
    t.trip_date > CURRENT_DATE
GROUP BY
```

```
    t.trip_id, t.country, t.trip_date, t.trip_name, t.max_no_places
HAVING
    (t.max_no_places - COUNT(r.status)) > 0;
```

## Zadanie 2 - funkcje

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

Procedury:

- `f_trip_participants`
  - zadaniem funkcji jest zwrócenie listy uczestników wskazanej wycieczki
  - parametry funkcji: `trip_id`
  - funkcja zwraca podobny zestaw danych jak widok `vw_eservation`
- `f_person_reservations`
  - zadaniem funkcji jest zwrócenie listy rezerwacji danej osoby
  - parametry funkcji: `person_id`
  - funkcja zwraca podobny zestaw danych jak widok `vw_reservation`
- `f_available_trips_to`
  - zadaniem funkcji jest zwrócenie listy wycieczek do wskazanego kraju, dostępnych w zadanym okresie czasu (od `date_from` do `date_to`)
  - parametry funkcji: `country`, `date_from`, `date_to`

Funkcje powinny zwracać tabelę/zbiór wynikowy. Należy rozważyć dodanie kontroli parametrów, (np. jeśli parametrem jest `trip_id` to można sprawdzić czy taka wycieczka istnieje). Podobnie jak w przypadku widoków należy zwrócić uwagę na strukturę kodu

Czy kontrola parametrów w przypadku funkcji ma sens?

- jakie są zalety/wady takiego rozwiązania?

Proponowany zestaw funkcji można rozbudować wedle uznania/potrzeb

- np. można dodać nowe/pomocnicze funkcje/procedury

## Zadanie 2 - rozwiązanie

```
create TYPE t_reservation AS OBJECT (
  reservation_id INT,
  trip_id INT,
  person_id INT,
  status CHAR
)

create type trip_participants as OBJECT
(
    FIRSTNAME varchar2(100),
    LASTNAME varchar2(100)
)

create function f_trip_participants(trip_id IN NUMBER)
    return trip_participants_table
as
    result trip_participants_table;
begin
    select trip_participants(p.FIRSTNAME, p.LASTNAME)
    bulk collect
    into result
    from person p
    inner join reservation r on p.person_id = r.person_id
    where r.trip_id = f_trip_participants.trip_id;

    return result;
end;

create FUNCTION f_person_reservations(p_person_id INT)
RETURN t_reservation_tab PIPELINED AS
BEGIN
  FOR r IN (SELECT * FROM reservation WHERE person_id = p_person_id) LOOP
    PIPE ROW(t_reservation(r.reservation_id, r.trip_id, r.person_id, r.status));
  END LOOP;
  RETURN;
END;

create FUNCTION f_available_trips_to(p_country VARCHAR2, p_date_from DATE, p_date_to DATE)
RETURN SYS_REFCURSOR AS
  v_cursor SYS_REFCURSOR;
BEGIN
  OPEN v_cursor FOR
  SELECT * FROM trip
  WHERE country = p_country AND trip_date BETWEEN p_date_from AND p_date_to;
  RETURN v_cursor;
END;
```

## Zadanie 3 - procedury

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

Procedury

- `p_add_reservation`
  - zadaniem procedury jest dopisanie nowej rezerwacji
  - parametry: `trip_id`, `person_id`,
  - procedura powinna kontrolować czy wycieczka jeszcze się nie odbyła, i czy sa wolne miejsca

    - procedura powinna również dopisywać inf. do tabeli `log`
- `p_modify_reservation_tatus`
    - zadaniem procedury jest zmiana statusu rezerwacji
    - parametry: `reservation_id`, `status`
    - procedura powinna 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)
    - procedura powinna również dopisywać inf. do tabeli `log`

Procedury:

- `p_modify_max_no_places`
    - zadaniem procedury jest zmiana maksymalnej liczby miejsc na daną wycieczkę
    - parametry: `trip_id`, `max_no_places`
    - 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 trip_id to należy sprawdzić czy taka wycieczka istnieje, jeśli robimy rezerwację to należy sprawdzać czy są wolne miejsca itp..)

Proponowany zestaw procedur można rozbudować wedle uznania/potrzeb

- np. można dodać nowe/pomocnicze funkcje/procedury

## Zadanie 3 - rozwiązanie

```
-- Procedura p_add_reservation
CREATE OR REPLACE PROCEDURE p_add_reservation(p_trip_id IN NUMBER, p_person_id IN NUMBER) AS
  v_max_no_places NUMBER;
  v_reserved_places NUMBER;
BEGIN
  SELECT max_no_places INTO v_max_no_places FROM trip WHERE trip_id = p_trip_id;
  SELECT COUNT(*) INTO v_reserved_places FROM reservation WHERE trip_id = p_trip_id AND status IN ('N', 'P');

  IF v_max_no_places > v_reserved_places THEN
    INSERT INTO reservation (trip_id, person_id, status) VALUES (p_trip_id, p_person_id, 'N');
    INSERT INTO log (reservation_id, log_date, status) VALUES (s_reservation_seq.currval, SYSDATE, 'N');
  ELSE
    RAISE_APPLICATION_ERROR(-20001, 'Brak wolnych miejsc na wycieczkę');
  END IF;
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    RAISE_APPLICATION_ERROR(-20002, 'Nie znaleziono wycieczki');
END;
/

-- Procedura p_modify_reservation_status
CREATE OR REPLACE PROCEDURE p_modify_reservation_status(p_reservation_id IN NUMBER, p_status IN CHAR) AS
  v_status CHAR(1);
BEGIN
  SELECT status INTO v_status FROM reservation WHERE reservation_id = p_reservation_id;

  IF v_status != 'C' OR (v_status = 'C' AND p_status != 'N') THEN
    UPDATE reservation SET status = p_status WHERE reservation_id = p_reservation_id;
    INSERT INTO log (reservation_id, log_date, status) VALUES (p_reservation_id, SYSDATE, p_status);
  ELSE
    RAISE_APPLICATION_ERROR(-20003, 'Nie można zmienić statusu anulowanej rezerwacji');
  END IF;
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    RAISE_APPLICATION_ERROR(-20004, 'Nie znaleziono rezerwacji');
END;
/

-- Procedura p_modify_max_no_places
CREATE OR REPLACE PROCEDURE p_modify_max_no_places(p_trip_id IN NUMBER, p_max_no_places IN NUMBER) AS
  v_reserved_places NUMBER;
BEGIN
  SELECT COUNT(*) INTO v_reserved_places FROM reservation WHERE trip_id = p_trip_id AND status IN ('N', 'P');

  IF p_max_no_places >= v_reserved_places THEN
    UPDATE trip SET max_no_places = p_max_no_places WHERE trip_id = p_trip_id;
  ELSE
    RAISE_APPLICATION_ERROR(-20005, 'Nie można zmniejszyć liczby miejsc poniżej liczby zarezerwowanych miejsc');
  END IF;
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    RAISE_APPLICATION_ERROR(-20006, 'Nie znaleziono wycieczki');
END;

Powyższe procedury używają transakcji do zapewnienia spójności danych. Jeśli wystąpi błąd, transakcja zostanie wycofana,
a dane pozostaną nienaruszone. Procedury te również sprawdzają, czy podane parametry są poprawne i czy operacje są dozwolone.
```

## Zadanie 4 - triggery

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

Triggery:

- trigger/triggery obsługujące
    - dodanie rezerwacji
    - zmianę statusu
- trigger zabraniający usunięcia rezerwacji

Oczywiście po wprowadzeniu tej zmiany należy "uaktualnić" procedury modyfikujące dane.

> UWAGA Należy stworzyć nowe wersje tych procedur (dodając do nazwy dopisek 4 - od numeru zadania). Poprzednie wersje procedur należy pozostawić w celu umożliwienia weryfikacji ich poprawności

Należy przygotować procedury: `p_add_reservation_4`, `p_modify_reservation_status_4`

## Zadanie 4 - rozwiązanie

```
-- Procedura p_add_reservation_4
CREATE OR REPLACE PROCEDURE p_add_reservation_4(p_trip_id IN NUMBER, p_person_id IN NUMBER) AS
  v_max_no_places NUMBER;
  v_reserved_places NUMBER;
BEGIN
  SELECT max_no_places INTO v_max_no_places FROM trip WHERE trip_id = p_trip_id;
  SELECT COUNT(*) INTO v_reserved_places FROM reservation WHERE trip_id = p_trip_id AND status IN ('N', 'P');

  IF v_max_no_places > v_reserved_places THEN
    INSERT INTO reservation (trip_id, person_id, status) VALUES (p_trip_id, p_person_id, 'N');
  ELSE
    RAISE_APPLICATION_ERROR(-20001, 'Brak wolnych miejsc na wycieczkę');
  END IF;
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    RAISE_APPLICATION_ERROR(-20002, 'Nie znaleziono wycieczki');
END;
/

-- Procedura p_modify_reservation_status_4
CREATE OR REPLACE PROCEDURE p_modify_reservation_status_4(p_reservation_id IN NUMBER, p_status IN CHAR) AS
  v_status CHAR(1);
BEGIN
  SELECT status INTO v_status FROM reservation WHERE reservation_id = p_reservation_id;

  IF v_status != 'C' OR (v_status = 'C' AND p_status != 'N') THEN
    UPDATE reservation SET status = p_status WHERE reservation_id = p_reservation_id;
  ELSE
    RAISE_APPLICATION_ERROR(-20003, 'Nie można zmienić statusu anulowanej rezerwacji');
  END IF;
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    RAISE_APPLICATION_ERROR(-20004, 'Nie znaleziono rezerwacji');
END;

-- Trigger obsługujący dodanie rezerwacji
CREATE OR REPLACE TRIGGER tr_add_reservation
AFTER INSERT ON reservation
FOR EACH ROW
BEGIN
  INSERT INTO log (reservation_id, log_date, status) VALUES (:new.reservation_id, SYSDATE, :new.status);
END;
/

-- Trigger obsługujący zmianę statusu
CREATE OR REPLACE TRIGGER tr_modify_reservation_status
AFTER UPDATE OF status ON reservation
FOR EACH ROW
BEGIN
  INSERT INTO log (reservation_id, log_date, status) VALUES (:new.reservation_id, SYSDATE, :new.status);
END;
/

-- Trigger zabraniający usunięcia rezerwacji
CREATE OR REPLACE TRIGGER tr_delete_reservation
BEFORE DELETE ON reservation
FOR EACH ROW
BEGIN
  RAISE_APPLICATION_ERROR(-20007, 'Nie można usunąć rezerwacji');
END;
/
```

## Zadanie 5 - triggery

Zmiana strategii kontroli dostępności miejsc. Realizacja przy pomocy triggerów

Należy wprowadzić zmianę, która spowoduje, że kontrola dostępności miejsc na wycieczki (przy dodawaniu nowej rezerwacji, zmianie statusu) będzie realizowana przy pomocy triggerów

Triggery:

- Trigger/triggery obsługujące:
  - dodanie rezerwacji
  - zmianę statusu

Oczywiście po wprowadzeniu tej zmiany należy "uaktualnić" procedury modyfikujące dane.

> UWAGA Należy stworzyć nowe wersje tych procedur (np. dodając do nazwy dopisek 5 - od numeru zadania). Poprzednie wersje procedur należy pozostawić w celu umożliwienia weryfikacji ich poprawności.

Należy przygotować procedury: `p_add_reservation_5`, `p_modify_reservation_status_5`

## Zadanie 5 - rozwiązanie

```
-- Procedura p_add_reservation_5
CREATE OR REPLACE PROCEDURE p_add_reservation_5(p_trip_id IN NUMBER, p_person_id IN NUMBER) AS
BEGIN
  INSERT INTO reservation (trip_id, person_id, status) VALUES (p_trip_id, p_person_id, 'N');
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    RAISE_APPLICATION_ERROR(-20003, 'Nie znaleziono wycieczki');
END;
/

-- Procedura p_modify_reservation_status_5
CREATE OR REPLACE PROCEDURE p_modify_reservation_status_5(p_reservation_id IN NUMBER, p_status IN CHAR) AS
BEGIN
```

```
  UPDATE reservation SET status = p_status WHERE reservation_id = p_reservation_id;
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    RAISE_APPLICATION_ERROR(-20004, 'Nie znaleziono rezerwacji');
END;
/

-- Trigger obsługujący dodanie rezerwacji
CREATE OR REPLACE TRIGGER tr_add_reservation_5
BEFORE INSERT ON reservation
FOR EACH ROW
DECLARE
  v_max_no_places NUMBER;
  v_reserved_places NUMBER;
BEGIN
  SELECT max_no_places INTO v_max_no_places FROM trip WHERE trip_id = :new.trip_id;
  SELECT COUNT(*) INTO v_reserved_places FROM reservation WHERE trip_id = :new.trip_id AND status IN ('N', 'P');

  IF v_max_no_places <= v_reserved_places THEN
    RAISE_APPLICATION_ERROR(-20001, 'Brak wolnych miejsc na wycieczkę');
  END IF;
END;
/

-- Trigger obsługujący zmianę statusu
CREATE OR REPLACE TRIGGER tr_modify_reservation_status_5
BEFORE UPDATE OF status ON reservation
FOR EACH ROW
DECLARE
  v_max_no_places NUMBER;
  v_reserved_places NUMBER;
BEGIN
  IF :old.status = 'C' AND :new.status = 'N' THEN
    SELECT max_no_places INTO v_max_no_places FROM trip WHERE trip_id = :new.trip_id;
    SELECT COUNT(*) INTO v_reserved_places FROM reservation WHERE trip_id = :new.trip_id AND status IN ('N', 'P');

    IF v_max_no_places <= v_reserved_places THEN
      RAISE_APPLICATION_ERROR(-20002, 'Brak wolnych miejsc na wycieczkę');
    END IF;
  END IF;
END;
/
```

## Zadanie 6

Zmiana struktury bazy danych. W tabeli `trip` należy dodać redundantne pole `no_available_places`. Dodanie redundantnego pola uprości kontrolę dostępnych miejsc, ale nieco skomplikuje procedury dodawania rezerwacji, zmiany statusu czy też zmiany maksymalnej liczby miejsc na wycieczki.

Należy przygotować polecenie/procedurę przeliczającą wartość pola `no_available_places` dla wszystkich wycieczek (do jednorazowego wykonania)

Obsługę pola `no_available_places` można zrealizować przy pomocy procedur lub triggerów

Należy zwrócić uwagę na spójność rozwiązania.

> UWAGA Należy stworzyć nowe wersje tych widoków/procedur/triggerów (np. dodając do nazwy dopisek 6 - od numeru zadania). Poprzednie wersje procedur należy pozostawić w celu umożliwienia weryfikacji ich poprawności.

- zmiana struktury tabeli

```
alter table trip add
    no_available_places int null
```

- polecenie przeliczające wartość `no_available_places`
  - należy wykonać operację "przeliczenia" liczby wolnych miejsc i aktualizacji pola `no_available_places`

## Zadanie 6 - rozwiązanie

```
CREATE OR REPLACE PROCEDURE p_calculate_no_available_places AS
BEGIN
  FOR t IN (SELECT trip_id, max_no_places FROM trip) LOOP
    UPDATE trip SET no_available_places = t.max_no_places - NVL((SELECT COUNT(*) FROM reservation WHERE trip_id = t.trip_id AND status IN ('N', 'P')), 0) WHERE trip_id =
t.trip_id;
  END LOOP;
END;
/

begin p_calculate_no_available_places(); end;

-- Triggery obsługujący dodanie rezerwacji
CREATE OR REPLACE TRIGGER tr_add_reservation_6_before_insert
BEFORE INSERT ON reservation
FOR EACH ROW
DECLARE
  v_max_no_places NUMBER;
BEGIN
  SELECT no_available_places INTO v_max_no_places FROM trip WHERE trip_id = :new.trip_id;

  IF v_max_no_places <= 0 THEN
    RAISE_APPLICATION_ERROR(-20001, 'Brak wolnych miejsc na wycieczkę');
  END IF;

  UPDATE trip SET no_available_places = no_available_places - 1 WHERE trip_id = :new.trip_id;
END;
/

CREATE OR REPLACE TRIGGER tr_add_reservation_6_after_insert
AFTER INSERT ON reservation
FOR EACH ROW
BEGIN
```

```
  INSERT INTO log (reservation_id, log_date, status) VALUES (:new.reservation_id, SYSDATE, :new.status);
END;
/

-- Triggery obsługujący zmianę statusu
CREATE OR REPLACE TRIGGER tr_modify_reservation_status_6_before_update
BEFORE UPDATE OF status ON reservation
FOR EACH ROW
DECLARE
  v_max_no_places NUMBER;
BEGIN
  IF :old.status = 'C' AND :new.status = 'N' THEN
    SELECT no_available_places INTO v_max_no_places FROM trip WHERE trip_id = :new.trip_id;

    IF v_max_no_places <= 0 THEN
      RAISE_APPLICATION_ERROR(-20002, 'Brak wolnych miejsc na wycieczkę');
    END IF;

    UPDATE trip SET no_available_places = no_available_places - 1 WHERE trip_id = :new.trip_id;
  ELSIF :old.status IN ('N', 'P') AND :new.status = 'C' THEN
    UPDATE trip SET no_available_places = no_available_places + 1 WHERE trip_id = :new.trip_id;
  END IF;
END;
/

CREATE OR REPLACE TRIGGER tr_modify_reservation_status_6_after_update
AFTER UPDATE OF status ON reservation
FOR EACH ROW
BEGIN
  INSERT INTO log (reservation_id, log_date, status) VALUES (:new.reservation_id, SYSDATE, :new.status);
END;
/

-- Procedura p_add_reservation_6
CREATE OR REPLACE PROCEDURE p_add_reservation_6(p_trip_id IN NUMBER, p_person_id IN NUMBER) AS
BEGIN
  INSERT INTO reservation (trip_id, person_id, status) VALUES (p_trip_id, p_person_id, 'N');
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    RAISE_APPLICATION_ERROR(-20003, 'Nie znaleziono wycieczki');
END;
/

-- Procedura p_modify_reservation_status_6
CREATE OR REPLACE PROCEDURE p_modify_reservation_status_6(p_reservation_id IN NUMBER, p_status IN CHAR) AS
BEGIN
  UPDATE reservation SET status = p_status WHERE reservation_id = p_reservation_id;
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    RAISE_APPLICATION_ERROR(-20004, 'Nie znaleziono rezerwacji');
END;
/
```

## Zadanie 6a - procedury

Obsługę pola `no_available_places` należy zrealizować przy pomocy procedur

- procedura dodająca rezerwację powinna aktualizować pole `no_available_places` w tabeli trip
- podobnie procedury odpowiedzialne za zmianę statusu oraz zmianę maksymalnej liczby miejsc na wycieczkę
- należy przygotować procedury oraz jeśli jest to potrzebne, zaktualizować triggery oraz widoki

> UWAGA Należy stworzyć nowe wersje tych widoków/procedur/triggerów (np. dodając do nazwy dopisek 6a - od numeru zadania). Poprzednie wersje procedur należy pozostawić w celu umożliwienia weryfikacji ich poprawności.

- może być potrzebne wyłączenie 'poprzednich wersji' triggerów

## Zadanie 6a - rozwiązanie

```
-- Procedura p_add_reservation_6a
CREATE OR REPLACE PROCEDURE p_add_reservation_6a(p_trip_id IN NUMBER, p_person_id IN NUMBER) AS
  v_max_no_places NUMBER;
BEGIN
  SELECT no_available_places INTO v_max_no_places FROM trip WHERE trip_id = p_trip_id;

  IF v_max_no_places <= 0 THEN
    RAISE_APPLICATION_ERROR(-20001, 'Brak wolnych miejsc na wycieczkę');
  END IF;

  INSERT INTO reservation (trip_id, person_id, status) VALUES (p_trip_id, p_person_id, 'N');
  UPDATE trip SET no_available_places = v_max_no_places - 1 WHERE trip_id = p_trip_id;

  INSERT INTO log (reservation_id, log_date, status) VALUES (s_reservation_seq.currval, SYSDATE, 'N');
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    RAISE_APPLICATION_ERROR(-20002, 'Nie znaleziono wycieczki');
END;
/

-- Procedura p_modify_reservation_status_6a
CREATE OR REPLACE PROCEDURE p_modify_reservation_status_6a(p_reservation_id IN NUMBER, p_status IN CHAR) AS
  v_status CHAR(1);
  v_trip_id NUMBER;
  v_max_no_places NUMBER;
BEGIN
  SELECT status, trip_id INTO v_status, v_trip_id FROM reservation WHERE reservation_id = p_reservation_id;

  IF v_status != 'C' AND p_status = 'C' THEN
    SELECT no_available_places INTO v_max_no_places FROM trip WHERE trip_id = v_trip_id;
    UPDATE trip SET no_available_places = v_max_no_places + 1 WHERE trip_id = v_trip_id;
  ELSIF v_status = 'C' AND p_status != 'C' THEN
    SELECT no_available_places INTO v_max_no_places FROM trip WHERE trip_id = v_trip_id;
```

```
    IF v_max_no_places <= 0 THEN
      RAISE_APPLICATION_ERROR(-20003, 'Brak wolnych miejsc na wycieczkę');
    END IF;

    UPDATE trip SET no_available_places = v_max_no_places - 1 WHERE trip_id = v_trip_id;
  END IF;

  UPDATE reservation SET status = p_status WHERE reservation_id = p_reservation_id;

  INSERT INTO log (reservation_id, log_date, status) VALUES (p_reservation_id, SYSDATE, p_status);
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    RAISE_APPLICATION_ERROR(-20004, 'Nie znaleziono rezerwacji');
END;
/

--ALTER TRIGGER tr_add_reservation_6_before_insert DISABLE;
--ALTER TRIGGER tr_add_reservation_6_after_insert DISABLE;
--ALTER TRIGGER tr_modify_reservation_status_6_before_update DISABLE;
--ALTER TRIGGER tr_modify_reservation_status_6_after_update DISABLE;

-- Widok vw_trip
CREATE OR REPLACE VIEW vw_trip_6a AS
SELECT
    trip_id,
    country,
    trip_date,
    trip_name,
    max_no_places,
    no_available_places
FROM
    trip;

-- Widok vw_available_trip
CREATE OR REPLACE VIEW vw_available_trip_6a AS
SELECT
    trip_id,
    country,
    trip_date,
    trip_name,
    max_no_places,
    no_available_places
FROM
    trip
WHERE
    trip_date > CURRENT_DATE AND no_available_places > 0;
```

## Zadanie 6b - triggery

Obsługę pola `no_available_places` należy zrealizować przy pomocy triggerów

- podczas dodawania rezerwacji trigger powinien aktualizować pole `no_available_places` w tabeli trip
- podobnie, podczas zmiany statusu rezerwacji
- należy przygotować trigger/triggery oraz jeśli jest to potrzebne, zaktualizować procedury modyfikujące dane oraz widoki

> UWAGA Należy stworzyć nowe wersje tych widoków/procedur/triggerów (np. dodając do nazwy dopisek 6b - od numeru zadania). Poprzednie wersje procedur należy pozostawić w celu umożliwienia weryfikacji ich poprawności.

- może być potrzebne wyłączenie 'poprzednich wersji' triggerów

## Zadanie 6b - rozwiązanie

```
-- Procedura p_add_reservation_6b
CREATE OR REPLACE PROCEDURE p_add_reservation_6b(p_trip_id IN NUMBER, p_person_id IN NUMBER) AS
BEGIN
  INSERT INTO reservation (trip_id, person_id, status) VALUES (p_trip_id, p_person_id, 'N');
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    RAISE_APPLICATION_ERROR(-20001, 'Nie znaleziono wycieczki');
END;
/

-- Procedura p_modify_reservation_status_6b
CREATE OR REPLACE PROCEDURE p_modify_reservation_status_6b(p_reservation_id IN NUMBER, p_status IN CHAR) AS
BEGIN
  UPDATE reservation SET status = p_status WHERE reservation_id = p_reservation_id;
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    RAISE_APPLICATION_ERROR(-20002, 'Nie znaleziono rezerwacji');
END;
/

-- Triggery obsługujący dodanie rezerwacji
CREATE OR REPLACE TRIGGER tr_add_reservation_6_before_insert
BEFORE INSERT ON reservation
FOR EACH ROW
DECLARE
  v_max_no_places NUMBER;
BEGIN
  SELECT no_available_places INTO v_max_no_places FROM trip WHERE trip_id = :new.trip_id;

  IF v_max_no_places <= 0 THEN
    RAISE_APPLICATION_ERROR(-20001, 'Brak wolnych miejsc na wycieczkę');
  END IF;

  UPDATE trip SET no_available_places = no_available_places - 1 WHERE trip_id = :new.trip_id;
END;
/
```

```
CREATE OR REPLACE TRIGGER tr_add_reservation_6_after_insert
AFTER INSERT ON reservation
FOR EACH ROW
BEGIN
  INSERT INTO log (reservation_id, log_date, status) VALUES (:new.reservation_id, SYSDATE, :new.status);
END;
/

-- Triggery obsługujący zmianę statusu
CREATE OR REPLACE TRIGGER tr_modify_reservation_status_6_before_update
BEFORE UPDATE OF status ON reservation
FOR EACH ROW
DECLARE
  v_max_no_places NUMBER;
BEGIN
  IF :old.status = 'C' AND :new.status = 'N' THEN
    SELECT no_available_places INTO v_max_no_places FROM trip WHERE trip_id = :new.trip_id;

    IF v_max_no_places <= 0 THEN
      RAISE_APPLICATION_ERROR(-20002, 'Brak wolnych miejsc na wycieczkę');
    END IF;

    UPDATE trip SET no_available_places = no_available_places - 1 WHERE trip_id = :new.trip_id;
  ELSIF :old.status IN ('N', 'P') AND :new.status = 'C' THEN
    UPDATE trip SET no_available_places = no_available_places + 1 WHERE trip_id = :new.trip_id;
  END IF;
END;
/

CREATE OR REPLACE TRIGGER tr_modify_reservation_status_6_after_update
AFTER UPDATE OF status ON reservation
FOR EACH ROW
BEGIN
  INSERT INTO log (reservation_id, log_date, status) VALUES (:new.reservation_id, SYSDATE, :new.status);
END;
/
```

## Zadanie 7 - podsumowanie

Porównaj sposób programowania w systemie Oracle PL/SQL ze znanym ci systemem/językiem MS Sqlserver T-SQL

```
Oracle PL/SQL i Microsoft SQL Server T-SQL to dwa różne języki służące do zarządzania bazami danych, które mają wiele podobieństw, ale także kilka kluczowych różnic. Oto kilka
z nich:

Podobieństwa:
Proceduralność: Obie technologie umożliwiają tworzenie procedur składowanych, funkcji, wyzwalaczy i pakietów, co umożliwia tworzenie bardziej złożonych i wydajnych operacji na
bazie danych.
Transakcyjność: Zarówno PL/SQL, jak i T-SQL obsługują transakcje, co pozwala na grupowanie wielu operacji na bazie danych w jedną jednostkę pracy.
Obsługa błędów: Obie technologie umożliwiają obsługę błędów za pomocą bloków TRY...CATCH (w T-SQL) lub BEGIN...EXCEPTION...END (w PL/SQL).

Różnice:
Składnia: Składnia w PL/SQL i T-SQL jest nieco inna. Na przykład, w PL/SQL używa się słowa kluczowego || do łączenia ciągów, podczas gdy w T-SQL używa się +.
Obsługa NULL: W T-SQL, porównanie czegokolwiek z NULL daje NULL, podczas gdy w PL/SQL, porównanie NULL z NULL daje prawdę.
Funkcje wbudowane: Obie technologie mają wiele funkcji wbudowanych, ale nie wszystkie są dostępne w obu. Na przykład, T-SQL ma funkcję GETDATE() do pobierania bieżącej daty i
godziny, podczas gdy PL/SQL używa SYSDATE.
Obsługa bloków kodu: W PL/SQL, bloki kodu są zdefiniowane za pomocą słów kluczowych DECLARE, BEGIN, EXCEPTION i END, podczas gdy w T-SQL używa się { i } do definiowania bloków
kodu.
```