

Oracle PL/Sql

widoki, funkcje, procedury, triggerzy ćwiczenie

Imiona i nazwiska autorów : Agnieszka Dąbek, Oliwia Dyszewska

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/bilety na wycieczkę
 - **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 Paryża', 'Francja', to_date('2023-09-12', 'YYYY-MM-DD'), 3);

insert into trip(trip_name, country, trip_date, max_no_places)
values ('Piękny Kraków', '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 zmodyfikować model danych tak żeby rezerwacja mogła dotyczyć kilku miejsc/biletów na wycieczkę

- do tabeli reservation należy dodać pole
 - no_tickets
- do tabeli log należy dodać pole
 - no_tickets

Należy zmodyfikować zestaw danych testowych

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

Transakcja to zbiór operacji na bazie danych, które muszą być wykonane w całości lub wcale.

Polecenie COMMIT zatwierdza zmiany, czyniąc je trwałymi, natomiast ROLLBACK cofa je do stanu początkowego.

W razie błędów, jeśli transakcja jest jawna (BEGIN TRANSACTION w T-SQL lub automatycznie rozpoczęta w PL/SQL), użytkownik może zdecydować o jej zatwierdzeniu lub wycofaniu.

W Oracle PL/SQL każda operacja rozpoczyna transakcję, którą trzeba jawnie zatwierdzić lub wycofać. Z kolei w MS SQL Server domyślnie działa autocommit, ale można zarządzać transakcjami jawnie. Obsługa błędów różni się: w Oracle PL/SQL stosuje się blok EXCEPTION, a w MS SQL Server TRY...CATCH.

pomocne mogą być materiały dostępne tu:

<https://upel.agh.edu.pl/mod/folder/view.php?id=311899> w szczególności

dokument: [1_oracle_modyf.pdf](#)

```
-- Modyfikacja tabeli reservation (dodanie pola no_tickets)
ALTER TABLE reservation ADD no_tickets INT DEFAULT 1;

-- Modyfikacja tabeli log (dodanie pola no_tickets)
ALTER TABLE log ADD no_tickets INT DEFAULT 1;

BEGIN
  -- Próba dodania nowej rezerwacji z określoną liczbą biletów
  INSERT INTO reservation(trip_id, person_id, status, no_tickets)
  VALUES (4, 9, 'P', 2);
  INSERT INTO reservation(trip_id, person_id, status, no_tickets)
  VALUES (2, 8, 'P', 2);
  INSERT INTO reservation(RESERVATION_ID, person_id, status, no_tickets)
  --próba wstawienia niedozwolonej wartości null
  VALUES (NULL, Default, 'P', 4);

  COMMIT;
EXCEPTION

  WHEN OTHERS THEN
    dbms_output.put_line('Błąd');
    raise;
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**, **no_tickets**
- **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, funkcje
- np. można zmienić def. widoków, dodając nowe/potrzebne pola

Zadanie 1 - rozwiązanie

```
--vw_reservation
--widok pokazujący dane wszystkich rezerwacji
SELECT
    r.reservation_id,
    t.country,
    t.trip_date,
    t.trip_name,
    p.firstname,
    p.lastname,
    r.status,
    t.trip_id,
    p.person_id,
    r.no_tickets
FROM
    reservation r
JOIN trip t ON r.trip_id = t.trip_id
JOIN person p ON r.person_id = p.person_id

--tabela utworzona przez widok vw_reservation
```

	RE...	COUNT...	TRIP_DATE	TRIP_NAME	FIRSTNAME	LASTNAME	STAT...	TRIP...	PERSON_ID	NO_TICKETS
1	161	Polska	2025-05-01	Hel	Jan	Nowak	N	1	1	1
2	33	Francja	2023-09-12	Wycieczka do Paryza	Jan	Nowak	P	2	1	4
3	32	Francja	2023-09-12	Wycieczka do Paryza	Jan	Kowalski	N	2	2	2
4	61	Polska	2025-05-01	Hel	Patryk	Kowalski	C	1	3	1
5	23	Francja	2023-09-12	Wycieczka do Paryza	Patryk	Kowalski	N	2	3	2

```
--vw_trip
--widok pokazujący dane o wszystkich wycieczkach
SELECT
    t.trip_id,
    t.country,
    t.trip_date,
    t.trip_name,
    t.max_no_places,
    t.max_no_places - NVL(r.total_tickets, 0) AS no_available_places
FROM
    trip t
LEFT JOIN (
    SELECT trip_id, SUM(no_tickets) AS total_tickets
    FROM reservation
    WHERE status IN ('N','P') -- rezerwacje aktywne
    GROUP BY trip_id
) r ON t.trip_id = r.trip_id
```

```
--tabela utworzona przez widok vw_trip
```

	TRIP_ID	COUNTRY	TRIP_DATE	TRIP_NAME	MAX_NO_PLACES	NO_AVAILABLE_PLACES
1	1	Polska	2025-05-01	Hel	70	24
2	2	Francja	2023-09-12	Wycieczka do Paryża	40	2
3	3	Polska	2025-05-03	Piekny Krakow	50	50
4	4	Francja	2025-11-17	Znow do Francji	20	13

```
--vw_available_trip
--widok pokazujący dane o wszystkich dostępnych wycieczkach
SELECT
    "TRIP_ID",
    "COUNTRY",
    "TRIP_DATE",
    "TRIP_NAME",
    "MAX_NO_PLACES",
    "NO_AVAILABLE_PLACES"
FROM vw_trip
WHERE trip_date > SYSDATE
    AND no_available_places > 0

--tabela utworzona przez widok vw_available_trip
```

	TRIP_ID	COUNTRY	TRIP_DATE	TRIP_NAME	MAX_NO_PLACES	NO_AVAILABLE_PLACES
1	1	Polska	2025-05-01	Hel	70	34
2	3	Polska	2025-05-03	Piekny Krakow	50	50
3	4	Francja	2025-11-17	Znow do Francji	20	13

```
--vw_n_reservations
--widok pokazujący dane wszystkich niepotwierdzonych rezerwacji
SELECT
    reservation_id,
    country,
    trip_date,
    trip_name,
    firstname,
    lastname,
    status,
    trip_id,
    person_id,
    no_tickets
FROM VW_RESERVATION
WHERE status = 'N'

--tabela utworzona przez widok vw_n_reservations
```


	RESERVATI...	COUNTRY	TRIP_DATE	TRIP_NAME	FIRSTNA...	LA...	ST...	TRIP...	PERSON...	NO_TICKETS
1	181	Polska	2025-05-01	Heł	Jan	Kowalski	N	1	2	3
2	32	Francja	2023-09-12	Wycieczka do Paryża	Jan	Kowalski	N	2	2	2
3	23	Francja	2023-09-12	Wycieczka do Paryża	Patryk	Kowalski	N	2	3	2
4	21	Francja	2023-09-12	Wycieczka do Paryża	Patryk	Kowalski	N	2	3	2
5	41	Francja	2023-09-12	Wycieczka do Paryża	Patryk	Kowalski	N	2	3	2

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_reservation`
- `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?

Kontrola parametrów w funkcjach ma sens, ponieważ zwiększa ich niezawodność i bezpieczeństwo.

Kontrola parametrów w funkcjach ma wiele zalet. Przede wszystkim zapobiega błędom wykonania, zapewniając, że funkcja otrzymuje poprawne dane. Poprawia także czytelność i przewidywalność kodu, ułatwiając debugowanie. Jasna walidacja parametrów sprawia również, że kod jest lepiej udokumentowany i bardziej intuicyjny.

Z drugiej strony, kontrola parametrów wiąże się z pewnymi wadami. Może powodować niewielki narzut wydajnościowy. Dodaje także złożoności do kodu, wymuszając dodatkowe warunki sprawdzające. W niektórych przypadkach może prowadzić do

powielania walidacji, jeśli sprawdzenie danych zostało już wykonane na wcześniejszym etapie.

Proponowany zestaw funkcji można rozbudować wedle uznania/potrzeb

- np. można dodać nowe/pomocnicze funkcje/procedury

Zadanie 2 - rozwiązanie

```
--f_trip_participants
create FUNCTION f_trip_participants(p_trip_id INT)
RETURN t_common_tab
AS
    v_result t_common_tab := t_common_tab();
BEGIN
    IF NOT f_trip_exists(p_trip_id) THEN
        RAISE_APPLICATION_ERROR(-20010, 'Wycieczka nie istnieje.');
```

END IF;

```
    SELECT t_common_rec(
        reservation_id, trip_id, person_id,
        firstname, lastname,
        status, no_tickets
    )
    BULK COLLECT INTO v_result
    FROM vw_reservation
    WHERE trip_id = p_trip_id;

    RETURN v_result;
END;
/

--przykładowe wywołanie funkcji
declare
    result      T_COMMON_TAB;
    P_TRIP_ID INT := 1;
begin
    result := BD_420909.F_TRIP_PARTICIPANTS(P_TRIP_ID => P_TRIP_ID);
    open ? for select * from table (result);
end;
--wynik
```

	RESERVATION_ID	TRIP_ID	PERSON_ID	FIRSTNAME	LASTNAME	STATUS	NO_TICKETS
1	161	1	1	Jan	Nowak	N	3
2	181	1	2	Jan	Kowalski	N	3
3	61	1	3	Patryk	Kowalski	C	1
4	201	1	3	Patryk	Kowalski	N	1
5	121	1	5	Kamil	Dul	N	1
6	31	1	5	Kamil	Dul	N	2
7	48	1	6	Zuzanna	Jedynak	P	8
8	141	1	6	Zuzanna	Jedynak	N	9
9	202	1	7	Jan	Grys	N	6
10	143	1	7	Jan	Grys	N	9
11	62	1	8	Ewa	Demarczyk	N	1
12	142	1	9	Damian	Gaska	P	10

```
--f_person_reservations
create FUNCTION f_person_reservations(p_person_id INT)
create FUNCTION f_person_reservations(p_person_id INT)
RETURN t_common_tab
AS
    v_result t_common_tab := t_common_tab();
    v_count NUMBER;
BEGIN
    SELECT COUNT(*) INTO v_count
    FROM person
    WHERE person_id = p_person_id;

    IF v_count = 0 THEN
        RAISE_APPLICATION_ERROR(-20002,
            'Osoba o ID = ' || p_person_id || ' nie istnieje w bazie.');
```

```
END IF;
SELECT t_common_rec(
    reservation_id,
    trip_id,
    person_id,
    firstname,
    lastname,
    status,
    no_tickets
)
    BULK COLLECT INTO v_result
    FROM vw_reservation
    WHERE person_id = p_person_id;

RETURN v_result;
END;
/

--przykładowe wywołanie funkcji
declare
    result      T_COMMON_TAB;
    P_PERSON_ID INT := 5;
begin
    result := BD_420909.F_PERSON_RESERVATIONS(P_PERSON_ID => P_PERSON_ID);
    open ? for select * from table (result);
```

```
end;
--wynik
```

	RESERVATION_ID	TRIP_ID	PERSON_ID	FIRSTNAME	LASTNAME	STATUS	NO_TICKETS
1	121	1	5	Kamil	Dul	N	1
2	31	1	5	Kamil	Dul	N	2

```
--f_available_trips_to
create FUNCTION f_available_trips_to(
    p_country    VARCHAR2,
    p_date_from  DATE,
    p_date_to    DATE
) RETURN t_available_trip_tab
AS
    v_result t_available_trip_tab := t_available_trip_tab();
BEGIN
    IF NOT f_country_exists(p_country) THEN
        RAISE_APPLICATION_ERROR(-20015, 'Nie ma żadnej wycieczki do kraju
'||p_country);
    END IF;
    SELECT t_available_trip_rec(
        trip_id,
        trip_name,
        country,
        trip_date,
        max_no_places,
        no_available_places
    )
    BULK COLLECT INTO v_result
    FROM vw_available_trip
    WHERE country      = p_country
        AND trip_date  BETWEEN p_date_from AND p_date_to
        AND no_available_places > 0;

    IF v_result.COUNT = 0 THEN
        RAISE_APPLICATION_ERROR(
            -20004,
            'W tym terminie nie ma dostępnych wycieczek do kraju: ' || p_country
        );
    END IF;

    RETURN v_result;
END;
/

--przykładowe wywołanie funkcji
declare
    result      T_AVAILABLE_TRIP_TAB;
    P_COUNTRY   VARCHAR2(4000) := 'Francja';
    P_DATE_FROM DATE           := TO_DATE('1900-01-01', 'YYYY-MM-DD');
```

```

P_DATE_TO    DATE                := TO_DATE('2100-01-01','YYYY-MM-DD');
begin
    result := BD_420909.F_AVAILABLE_TRIPS_TO(
        P_COUNTRY => P_COUNTRY,
        P_DATE_FROM => P_DATE_FROM,
        P_DATE_TO => P_DATE_TO
    );
    open ? for select * from table (result);
end;
--wynik

```

TRIP_ID	TRIP_NAME	COUNTRY	TRIP_DATE	MAX_NO_PLACES	NO_AVAILABLE_PLACES
4	Znow do Francji	Francja	2025-11-17	20	3

```

--funkcje pomocnicze

--f_trip_exists
--sprawdza czy wycieczka o podanym id istnieje
create FUNCTION f_trip_exists(p_trip_id INT)
RETURN BOOLEAN
AS
    exist NUMBER;
BEGIN
    SELECT CASE
        WHEN EXISTS (SELECT 1 FROM trip WHERE trip_id = p_trip_id) THEN 1
        ELSE 0
    END
    INTO exist
    FROM dual;

    IF exist = 1 THEN
        RETURN TRUE;
    ELSE
        RETURN FALSE;
    END IF;
END;
/

--f_trip_available
--sprawdza czy wycieczka o podanym id jest dostępna pod względem ilości wolnych
miejsc
create FUNCTION f_trip_available(p_trip_id INT)
RETURN BOOLEAN
AS
    exist NUMBER;
BEGIN
    SELECT CASE
        WHEN EXISTS (SELECT 1 FROM vw_available_trip WHERE trip_id =
p_trip_id AND no_available_places > 0) THEN 1
        ELSE 0
    END
    INTO exist

```

```
FROM dual;

IF exist = 1 THEN
    RETURN TRUE;
ELSE
    RETURN FALSE;
END IF;
END;
/

--f_person_exists
--sprawdza czy osoba o podanym id istnieje
create FUNCTION f_person_exists(p_person_id INT)
RETURN BOOLEAN
AS
    exist NUMBER;
BEGIN
    SELECT CASE
        WHEN EXISTS (SELECT 1 FROM person WHERE person_id = p_person_id) THEN
1
            ELSE 0
        END
    INTO exist
    FROM dual;

    IF exist = 1 THEN
        RETURN TRUE;
    ELSE
        RETURN FALSE;
    END IF;
END;
/

--f_reservation_exists
--sprawdza czy rezerwacja o podanym id istnieje
create FUNCTION f_reservation_exists(p_reservation_id INT)
RETURN BOOLEAN
AS
    exist NUMBER;
BEGIN
    SELECT CASE
        WHEN EXISTS (SELECT 1 FROM reservation WHERE reservation_id =
p_reservation_id) THEN 1
            ELSE 0
        END
    INTO exist
    FROM dual;

    IF exist = 1 THEN
        RETURN TRUE;
    ELSE
        RETURN FALSE;
    END IF;
END;
```

```
/
--f_country_exists
--sprawdza czy do kraju o podanej nazwie można pojechać na wycieczkę (lub odbyła
się do niego jakaś wycieczka w przeszłości)
create FUNCTION f_country_exists(p_country TRIP.COUNTRY%TYPE)
    RETURN BOOLEAN
AS
    exist NUMBER;
BEGIN
    SELECT CASE
        WHEN EXISTS (SELECT * FROM trip WHERE country = p_country) THEN 1
        ELSE 0
    END
    INTO exist
    FROM dual;

    IF exist = 1 THEN
        RETURN TRUE;
    ELSE
        RETURN FALSE;
    END IF;
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**, **no_tickets**
 - procedura powinna kontrolować czy wycieczka jeszcze się nie odbyła, i czy są wolne miejsca
 - procedura powinna również dopisywać inf. do tabeli **log**
- **p_modify_reservation_status**
 - 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**

- **p_modify_reservation**
 - zadaniem procedury jest zmiana statusu rezerwacji
 - parametry: **reservation_id**, **no_tickets**
 - procedura powinna kontrolować czy możliwa jest zmiana liczby sprzedanych/zarezerwowanych biletów – może już nie być miejsc
 - procedura powinna również dopisywać inf. do tabeli **log**
- **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

```
--p_add_reservation
create PROCEDURE p_add_reservation(
    p_trip_id    IN INT,
    p_person_id  IN INT,
    p_no_tickets IN INT
) AS
    v_trip_date    DATE;
    v_no_available INT;
BEGIN

    SELECT trip_date, no_available_places
    INTO v_trip_date, v_no_available
    FROM vw_trip
    WHERE trip_id = p_trip_id;

    IF v_trip_date <= SYSDATE THEN
        RAISE_APPLICATION_ERROR(-20001, 'Wycieczka już się odbyła.');
```

```
END IF;

    IF p_no_tickets > v_no_available THEN
        RAISE_APPLICATION_ERROR(-20002, 'Brak wystarczającej liczby miejsc.');
```

```
END IF;
```



```

INSERT INTO reservation(trip_id, person_id, status, no_tickets)
VALUES (p_trip_id, p_person_id, 'N', p_no_tickets);

INSERT INTO log(reservation_id, log_date, status, no_tickets)
VALUES (s_reservation_seq.currval, SYSDATE, 'N', p_no_tickets);

EXCEPTION
    WHEN NO_DATA_FOUND THEN
        RAISE_APPLICATION_ERROR(-20010, 'Nie znaleziono wycieczki o
ID='||p_trip_id);
    WHEN OTHERS THEN
        ROLLBACK;
        RAISE;
END;
/

--przykładowe użycie procedury
--dodanie rezerwacji na wycieczkę o id 1 dla osoby o id 5 z liczbą biletów równą 1
declare
    P_TRIP_ID      INT := 1;
    P_PERSON_ID    INT := 5;
    P_NO_TICKETS   INT := 1;
begin
    BD_420909.P_ADD_RESERVATION(
        P_TRIP_ID => P_TRIP_ID,
        P_PERSON_ID => P_PERSON_ID,
        P_NO_TICKETS => P_NO_TICKETS
    );
end;

--p_modify_reservation_status
create PROCEDURE p_modify_reservation_status(
    p_reservation_id IN INT,
    p_status         IN VARCHAR2
) IS
    v_current_status VARCHAR2(1);
BEGIN
    SELECT status INTO v_current_status
    FROM reservation
    WHERE reservation_id = p_reservation_id;

    IF v_current_status = 'C' THEN
        RAISE_APPLICATION_ERROR(-20003, 'Nie można zmienić statusu anulowanej
rezerwacji.');
```

```

    END IF;

    UPDATE reservation
    SET status = p_status
    WHERE reservation_id = p_reservation_id;

    INSERT INTO log(reservation_id, log_date, status, no_tickets)
    VALUES (p_reservation_id, SYSDATE, p_status,
            (SELECT no_tickets FROM reservation WHERE reservation_id =
```

```
p_reservation_id));

EXCEPTION
    WHEN OTHERS THEN
        ROLLBACK;
        RAISE;
END;
/

--przykładowe użycie procedury
--zmiana statusu rezerwacji o numerze 121 na 'P'
declare
    P_RESERVATION_ID INT          := 121;
    P_STATUS          VARCHAR2(4000) := 'P';
begin
    BD_420909.P_MODIFY_RESERVATION_STATUS(
        P_RESERVATION_ID => P_RESERVATION_ID,
        P_STATUS => P_STATUS
    );
end;

--p_modify_reservation
create PROCEDURE p_modify_reservation(
    p_reservation_id IN INT,
    p_no_tickets     IN INT
) AS
    v_trip_id          INT;
    v_current_no_tickets INT;
    v_max_no_places    INT;
    v_reserved         INT;
BEGIN
    SELECT trip_id, no_tickets INTO v_trip_id, v_current_no_tickets
    FROM reservation
    WHERE reservation_id = p_reservation_id;

    SELECT max_no_places INTO v_max_no_places
    FROM trip
    WHERE trip_id = v_trip_id;

    SELECT NVL(SUM(no_tickets),0) INTO v_reserved
    FROM reservation
    WHERE trip_id = v_trip_id AND reservation_id <> p_reservation_id AND status IN
('N','P');

    IF (v_reserved + p_no_tickets) > v_max_no_places THEN
        RAISE_APPLICATION_ERROR(-20004, 'Zmiana liczby biletów niemożliwa -
przekroczono limit miejsc.');
```

```
        END IF;

        UPDATE reservation
        SET no_tickets = p_no_tickets
        WHERE reservation_id = p_reservation_id;
```

```

    INSERT INTO log(reservation_id, log_date, status, no_tickets)
    VALUES (p_reservation_id, SYSDATE, (SELECT status FROM reservation WHERE
reservation_id = p_reservation_id), p_no_tickets);

EXCEPTION
    WHEN OTHERS THEN
        ROLLBACK;
        RAISE;

END;
/

--przykładowe użycie procedury
--zmiana ilości biletów rezerwacji o id 121 na 2
declare
    P_RESERVATION_ID INT := 121;
    P_NO_TICKETS      INT := 2;
begin
    BD_420909.P_MODIFY_RESERVATION(
        P_RESERVATION_ID => P_RESERVATION_ID,
        P_NO_TICKETS => P_NO_TICKETS
    );
end;

--p_modify_max_no_places
create PROCEDURE p_modify_max_no_places(
    p_trip_id      IN INT,
    p_max_no_places IN INT
) AS
    v_reserved INT;
BEGIN
    SELECT NVL(SUM(no_tickets),0) INTO v_reserved
    FROM reservation
    WHERE trip_id = p_trip_id AND status IN ('N','P');

    IF p_max_no_places < v_reserved THEN
        RAISE_APPLICATION_ERROR(-20005, 'Nowa maksymalna liczba miejsc jest
mniejsza niż liczba już zarezerwowanych.');
```

END IF;

```

    UPDATE trip
    SET max_no_places = p_max_no_places
    WHERE trip_id = p_trip_id;

EXCEPTION
    WHEN OTHERS THEN
        ROLLBACK;
        RAISE;

END;
/

--przykładowe użycie procedury
--zwiększenie ilości miejsc na wycieczkę o id 1 do 60 miejsc
declare
```

```
P_TRIP_ID      INT := 1;
P_MAX_NO_PLACES INT := 60;
begin
  BD_420909.P_MODIFY_MAX_NO_PLACES(
    P_TRIP_ID => P_TRIP_ID,
    P_MAX_NO_PLACES => P_MAX_NO_PLACES
  );
end;
```

Zadanie 4 - triggery

Zmiana strategii zapisywania do dziennika rezerwacji. Realizacja przy pomocy triggerów

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

Triggery:

- trigger/triggery obsługujące
 - dodanie rezerwacji
 - zmianę statusu
 - zmianę liczby zarezerwowanych/kupionych biletów
- 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`, `p_modify_reservation_4`

Zadanie 4 - rozwiązanie

```
--trigger TRG_RESERVATION_LOG
--obsługuje dodanie rezerwacji
create trigger TRG_RESERVATION_LOG
  after insert
  on RESERVATION
  for each row
BEGIN
  INSERT INTO log(reservation_id, log_date, status, no_tickets)
```

```
VALUES (:NEW.reservation_id, SYSDATE, :NEW.status, :NEW.no_tickets);
END;
/

--trigger TRG_PREVENT_RESERVATION_DELETE
--zabrania usunięcia rezerwacji
create trigger TRG_PREVENT_RESERVATION_DELETE
before delete
on RESERVATION
for each row
BEGIN
RAISE_APPLICATION_ERROR(-20006, 'Usunięcie rezerwacji jest niedozwolone.');
```

```
END;
/

--trigger TRG_LOG_RESERVATION_UPDATE
--obsługuje zmianę statusu rezerwacji i liczby biletów/miejsc
create trigger TRG_LOG_RESERVATION_UPDATE
after update of STATUS,NO_TICKETS
on RESERVATION
for each row
BEGIN
INSERT INTO log(reservation_id, log_date, status, no_tickets)
VALUES (:NEW.reservation_id, SYSDATE, :NEW.status, :NEW.no_tickets);
END;
/

--nowe wersje procedur
--p_add_reservation_4
create PROCEDURE p_add_reservation_4(
p_trip_id IN INT,
p_person_id IN INT,
p_no_tickets IN INT
) IS
v_trip_date DATE;
v_max_places INT;
v_reserved INT;
BEGIN
IF NOT f_trip_exists(p_trip_id) THEN
RAISE_APPLICATION_ERROR(-20010, 'Wycieczka nie istnieje.');
```

```
END IF;
IF NOT f_person_exists(p_person_id) THEN
RAISE_APPLICATION_ERROR(-20011, 'Osoba nie istnieje.');
```

```
END IF;

SELECT trip_date, max_no_places
INTO v_trip_date, v_max_places
FROM trip
WHERE trip_id = p_trip_id;

IF v_trip_date <= SYSDATE THEN
RAISE_APPLICATION_ERROR(-20001, 'Wycieczka już się odbyła.');
```

```
END IF;
```

```
SELECT NVL(SUM(no_tickets),0)
      INTO v_reserved
      FROM reservation
      WHERE trip_id = p_trip_id AND status IN ('N','P');

IF v_reserved + p_no_tickets > v_max_places THEN
  RAISE_APPLICATION_ERROR(-20002, 'Brak dostępnych miejsc.');
```

END IF;

```
INSERT INTO reservation(trip_id, person_id, status, no_tickets)
VALUES (p_trip_id, p_person_id, 'N', p_no_tickets);

EXCEPTION
  WHEN OTHERS THEN
    ROLLBACK;
    RAISE;

END;
/

--przykładowe użycie procedury
--dodanie rezerwacji na wycieczkę o id 1 dla osoby o id 5 z liczbą biletów równą 1
declare
  P_TRIP_ID      INT := 1;
  P_PERSON_ID    INT := 5;
  P_NO_TICKETS   INT := 1;
begin
  BD_420909.P_ADD_RESERVATION_4(
    P_TRIP_ID => P_TRIP_ID,
    P_PERSON_ID => P_PERSON_ID,
    P_NO_TICKETS => P_NO_TICKETS
  );
end;

--p_modify_reservation_status_4
create PROCEDURE p_modify_reservation_status_4(
  p_reservation_id IN INT,
  p_status         IN VARCHAR2
) IS
  v_current_status VARCHAR2(1);
BEGIN
  IF NOT f_reservation_exists(p_reservation_id) THEN
    RAISE_APPLICATION_ERROR(-20010, 'Rezerwacja nie istnieje.');
```

END IF;

```
SELECT status INTO v_current_status
FROM reservation
WHERE reservation_id = p_reservation_id;

IF v_current_status = 'C' THEN
  RAISE_APPLICATION_ERROR(-20003, 'Nie można zmienić statusu anulowanej
```

```
rezerwacji.');
```

```
    END IF;
```

```
    UPDATE reservation
      SET status = p_status
    WHERE reservation_id = p_reservation_id;
```

```
EXCEPTION
  WHEN OTHERS THEN
    ROLLBACK;
    RAISE;
```

```
END;
```

```
/
```

```
--przykładowe użycie procedury
--zmiana statusu rezerwacji o numerze 121 na 'P'
```

```
declare
  P_RESERVATION_ID INT          := 121;
  P_STATUS          VARCHAR2(4000) := 'P';
begin
  BD_420909.P_MODIFY_RESERVATION_STATUS_4(
    P_RESERVATION_ID => P_RESERVATION_ID,
    P_STATUS => P_STATUS
  );
end;
```

```
--p_modify_reservation_4
create PROCEDURE p_modify_reservation_4(
  p_reservation_id IN INT,
  p_no_tickets     IN INT
) IS
  v_trip_id     INT;
  v_max_places  INT;
  v_reserved    INT;
BEGIN
  IF NOT f_reservation_exists(p_reservation_id) THEN
    RAISE_APPLICATION_ERROR(-20010, 'Rezerwacja nie istnieje.');
```

```
  END IF;
```

```
  SELECT trip_id INTO v_trip_id
    FROM reservation
   WHERE reservation_id = p_reservation_id;
```

```
  SELECT max_no_places INTO v_max_places
    FROM trip
   WHERE trip_id = v_trip_id;
```

```
  SELECT NVL(SUM(no_tickets),0)
    INTO v_reserved
   FROM reservation
  WHERE trip_id = v_trip_id
    AND reservation_id <> p_reservation_id
    AND status IN ('N','P');
```

```
IF v_reserved + p_no_tickets > v_max_places THEN
    RAISE_APPLICATION_ERROR(-20004, 'Brak dostępnych miejsc przy zmianie liczby
biletów.');
```

```
END IF;

UPDATE reservation
    SET no_tickets = p_no_tickets
    WHERE reservation_id = p_reservation_id;

EXCEPTION
    WHEN OTHERS THEN
        ROLLBACK;
        RAISE;
END;
/

--przykładowe użycie procedury
--zmiana ilości biletów rezerwacji o id 121 na 2
declare
    P_RESERVATION_ID INT := 121;
    P_NO_TICKETS      INT := 2;
begin
    BD_420909.P_MODIFY_RESERVATION_4(
        P_RESERVATION_ID => P_RESERVATION_ID,
        P_NO_TICKETS => P_NO_TICKETS
    );
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
 - zmianę liczby zakupionych/zarezerwowanych miejsc/biletów

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`, `p_modify_reservation_status_5`

Zadanie 5 - rozwiązanie

```
--trigger TRG_RESERVATION_CHECK_INSERT_CT
--obsługuje dodanie rezerwacji
create trigger TRG_RESERVATION_CHECK_INSERT_CT
  instead of insert
  on RESERVATION
  for each row
  COMPOUND TRIGGER

  TYPE t_insert_row IS RECORD (
    trip_id      INT,
    person_id    INT,
    no_tickets    INT
  );
  TYPE t_insert_row_tab IS TABLE OF t_insert_row INDEX BY PLS_INTEGER;

  g_rows  t_insert_row_tab;
  g_count PLS_INTEGER := 0;

  BEFORE EACH ROW IS
  BEGIN
    IF NOT f_trip_exists(:NEW.trip_id) THEN
      RAISE_APPLICATION_ERROR(-20001, 'Wycieczki nie znaleziono');
    END IF;

    IF NOT f_person_exists(:NEW.person_id) THEN
      RAISE_APPLICATION_ERROR(-20002, 'Osoby nie znaleziono');
    END IF;

    IF :NEW.no_tickets < 1 THEN
      RAISE_APPLICATION_ERROR(-20003, 'no_tickets < 1');
    END IF;

    g_count := g_count + 1;
    g_rows(g_count).trip_id      := :NEW.trip_id;
    g_rows(g_count).person_id    := :NEW.person_id;
    g_rows(g_count).no_tickets   := :NEW.no_tickets;

  END BEFORE EACH ROW;

  AFTER STATEMENT IS
  v_available_places INT;
  BEGIN
    FOR i IN 1..g_count LOOP
```

```
IF NOT f_trip_available(g_rows(i).trip_id) THEN
    RAISE_APPLICATION_ERROR(-20004, 'Wycieczka niedostępna.');
```

END IF;

```
BEGIN
    SELECT no_available_places
        INTO v_available_places
        FROM vw_available_trip
        WHERE trip_id = g_rows(i).trip_id;

    IF v_available_places < g_rows(i).no_tickets THEN
        RAISE_APPLICATION_ERROR(-20005, 'Nie ma tylu wolnych miejsc');
```

END IF;

```
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        RAISE_APPLICATION_ERROR(-20006, 'Nie znaleziono danych');
```

END;

END LOOP;

g_rows.DELETE;

g_count := 0;

END AFTER STATEMENT;

END trg_reservation_check_insert_ct;

/

--trigger TRG_RESERVATION_CHECK_STATUS_UPDATE_CT

--obsługuje aktualizację statusu rezerwacji

create trigger TRG_RESERVATION_CHECK_STATUS_UPDATE_CT

instead of update of STATUS

on RESERVATION

for each row

COMPOUND TRIGGER

TYPE t_status_change IS RECORD (

old_status CHAR(1),

new_status CHAR(1),

trip_id INT,

no_tickets INT

);

TYPE t_status_change_tab IS TABLE OF t_status_change INDEX BY PLS_INTEGER;

g_changes t_status_change_tab;

g_count PLS_INTEGER := 0;

BEFORE EACH ROW IS

BEGIN

g_count := g_count + 1;

g_changes(g_count).old_status := :OLD.status;

g_changes(g_count).new_status := :NEW.status;

g_changes(g_count).trip_id := :OLD.trip_id;

g_changes(g_count).no_tickets := :OLD.no_tickets;

END BEFORE EACH ROW;

```
AFTER STATEMENT IS
v_available_places INT;
BEGIN
FOR i IN 1..g_count LOOP
    IF g_changes(i).old_status = 'C' AND g_changes(i).new_status IN ('N','P') THEN
        IF NOT f_trip_available(g_changes(i).trip_id) THEN
            RAISE_APPLICATION_ERROR(-20003, 'Wycieczka niedostępna');
        END IF;

        SELECT no_available_places
            INTO v_available_places
            FROM vw_available_trip
            WHERE trip_id = g_changes(i).trip_id;

        IF v_available_places < g_changes(i).no_tickets THEN
            RAISE_APPLICATION_ERROR(-20004, 'Zbyt mało dostępnych miejsc');
        END IF;
    END IF;
END LOOP;

g_changes.DELETE;
g_count := 0;
END AFTER STATEMENT;

END trg_reservation_check_status_update_ct;
/

--trigger TRG_CHECK_RESERVATION_TICKETS_CT
--obsługuje aktualizację ilości biletów/miejsc w rezerwacji
create trigger TRG_CHECK_RESERVATION_TICKETS_CT
    instead of update of NO_TICKETS
    on RESERVATION
    for each row
    COMPOUND TRIGGER

    TYPE t_tickets_change IS RECORD(
        old_tickets INT,
        new_tickets INT,
        trip_id     INT
    );
    TYPE t_tickets_change_tab IS TABLE OF t_tickets_change INDEX BY PLS_INTEGER;

    g_tickets  t_tickets_change_tab;
    g_count    PLS_INTEGER := 0;

    BEFORE EACH ROW IS
    BEGIN
        IF :NEW.no_tickets < 1 THEN
            RAISE_APPLICATION_ERROR(-20001, 'Liczba biletów musi być >= 1');
        END IF;

        g_count := g_count + 1;
        g_tickets(g_count).old_tickets := :OLD.no_tickets;
        g_tickets(g_count).new_tickets := :NEW.no_tickets;
```

```

        g_tickets(g_count).trip_id      := :OLD.trip_id;
    END BEFORE EACH ROW;

    AFTER STATEMENT IS
    v_available_places INT;
    v_diff INT;
BEGIN
    FOR i IN 1..g_count LOOP
        IF g_tickets(i).new_tickets > g_tickets(i).old_tickets THEN
            SELECT no_available_places
            INTO v_available_places
            FROM vw_available_trip
            WHERE trip_id = g_tickets(i).trip_id;

            v_diff := g_tickets(i).new_tickets - g_tickets(i).old_tickets;
            IF v_diff > v_available_places THEN
                RAISE_APPLICATION_ERROR(-20003, 'Zbyt mało dostępnych miejsc');
            END IF;
        END IF;
    END LOOP;

    g_tickets.DELETE;
    g_count := 0;
END AFTER STATEMENT;

END trg_check_reservation_tickets_ct;
/

--nowe wersje procedur
--p_add_reservation_5
create PROCEDURE p_add_reservation_5(
    p_trip_id    IN INT,
    p_person_id  IN INT,
    p_no_tickets IN INT
) AS
BEGIN
    INSERT INTO reservation(trip_id, person_id, status, no_tickets)
    VALUES (p_trip_id, p_person_id, 'N', p_no_tickets);
    COMMIT;
END;
/

--przykładowe użycie procedury
--dodanie rezerwacji na wycieczkę o id 1 dla osoby o id 5 z liczbą biletów równą 1
declare
    P_TRIP_ID    INT := 1;
    P_PERSON_ID  INT := 5;
    P_NO_TICKETS INT := 1;
begin
    BD_420909.P_ADD_RESERVATION_5(
        P_TRIP_ID => P_TRIP_ID,
        P_PERSON_ID => P_PERSON_ID,
        P_NO_TICKETS => P_NO_TICKETS

```

```
);
end;

--p_modify_reservation_status_5
create PROCEDURE p_modify_reservation_status_5(
    p_reservation_id IN INT,
    p_status          IN CHAR
) AS
BEGIN
    UPDATE reservation
        SET status = p_status
        WHERE reservation_id = p_reservation_id;
    COMMIT;
END;
/

--przykładowe użycie procedury
--zmiana statusu rezerwacji o numerze 121 na 'P'
declare
    P_RESERVATION_ID INT      := 121;
    P_STATUS          CHAR(50) := 'P';
begin
    BD_420909.P_MODIFY_RESERVATION_STATUS_5(
        P_RESERVATION_ID => P_RESERVATION_ID,
        P_STATUS => P_STATUS
    );
end;

--p_modify_reservation_5
create PROCEDURE p_modify_reservation_5(
    p_reservation_id IN INT,
    p_no_tickets     IN INT
) AS
BEGIN
    UPDATE reservation
        SET no_tickets = p_no_tickets
        WHERE reservation_id = p_reservation_id;
    COMMIT;
END;
/

--przykładowe użycie procedury
--zmiana ilości biletów rezerwacji o id 121 na 2
declare
    P_RESERVATION_ID INT := 121;
    P_NO_TICKETS     INT := 2;
begin
    BD_420909.P_MODIFY_RESERVATION_5(
        P_RESERVATION_ID => P_RESERVATION_ID,
        P_NO_TICKETS => P_NO_TICKETS
    );
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

```
--procedura przeliczajaca wartość pola `no_available_places` dla wszystkich  
wycieczek  
create PROCEDURE recalc_no_available_places IS  
BEGIN  
    UPDATE trip t  
    SET no_available_places = t.max_no_places - NVL(  
        (SELECT SUM(no_tickets)  
        FROM reservation r  
        WHERE r.trip_id = t.trip_id AND status IN ('N','P')), 0);  
    COMMIT;  
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 triggerów 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

```
--nowe wersje procedur
--p_add_reservation_6a
create PROCEDURE p_add_reservation_6a(
    p_trip_id    IN INT,
    p_person_id  IN INT,
    p_no_tickets IN INT
) AS
    v_trip_date DATE;
BEGIN

    SELECT trip_date
    INTO v_trip_date
    FROM trip
    WHERE trip_id = p_trip_id;

    INSERT INTO reservation(trip_id, person_id, status, no_tickets)
    VALUES (p_trip_id, p_person_id, 'N', p_no_tickets);

    UPDATE trip
    SET no_available_places = no_available_places - p_no_tickets
    WHERE trip_id = p_trip_id;

EXCEPTION
    WHEN OTHERS THEN
        ROLLBACK;
```

```
        RAISE;
    END;
/

--przykładowe użycie procedury
--dodanie rezerwacji na wycieczkę o id 1 dla osoby o id 5 z liczbą biletów równą 1
declare
    P_TRIP_ID      INT := 1;
    P_PERSON_ID    INT := 5;
    P_NO_TICKETS   INT := 1;
begin
    BD_420909.P_ADD_RESERVATION_6A(
        P_TRIP_ID => P_TRIP_ID,
        P_PERSON_ID => P_PERSON_ID,
        P_NO_TICKETS => P_NO_TICKETS
    );
end;

--p_modify_reservation_status_6a
create PROCEDURE p_modify_reservation_status_6a(
    p_reservation_id IN INT,
    p_status         IN VARCHAR2
) AS
    v_current_status VARCHAR2(1);
    v_trip_id         INT;
    v_no_tickets       INT;
BEGIN
    IF NOT f_reservation_exists(p_reservation_id) THEN
        RAISE_APPLICATION_ERROR(-20012, 'Rezerwacja nie istnieje.');
```

```
    END IF;

    SELECT status, trip_id, no_tickets
    INTO v_current_status, v_trip_id, v_no_tickets
    FROM reservation
    WHERE reservation_id = p_reservation_id;

    IF v_current_status = p_status THEN
        RETURN;
    END IF;

    UPDATE reservation
    SET status = p_status
    WHERE reservation_id = p_reservation_id;

    IF v_current_status IN ('N','P') AND p_status = 'C' THEN
        UPDATE trip
        SET no_available_places = no_available_places + v_no_tickets
        WHERE trip_id = v_trip_id;
    ELSIF v_current_status = 'C' AND p_status IN ('N','P') THEN
        UPDATE trip
        SET no_available_places = no_available_places - v_no_tickets
        WHERE trip_id = v_trip_id;
    END IF;
```



```
EXCEPTION
    WHEN OTHERS THEN
        ROLLBACK;
        RAISE;
END;
/

--przykładowe użycie procedury
--zmiana statusu rezerwacji o numerze 121 na 'P'
declare
    P_RESERVATION_ID INT          := 121;
    P_STATUS          VARCHAR2(4000) := 'P';
begin
    BD_420909.P_MODIFY_RESERVATION_STATUS_6A(
        P_RESERVATION_ID => P_RESERVATION_ID,
        P_STATUS => P_STATUS
    );
end;

--p_modify_reservation_6a
create PROCEDURE p_add_reservation_6a(
    p_trip_id    IN INT,
    p_person_id  IN INT,
    p_no_tickets IN INT
) AS
    v_trip_date DATE;
BEGIN

    SELECT trip_date
    INTO v_trip_date
    FROM trip
    WHERE trip_id = p_trip_id;

    INSERT INTO reservation(trip_id, person_id, status, no_tickets)
    VALUES (p_trip_id, p_person_id, 'N', p_no_tickets);

    UPDATE trip
    SET no_available_places = no_available_places - p_no_tickets
    WHERE trip_id = p_trip_id;

EXCEPTION
    WHEN OTHERS THEN
        ROLLBACK;
        RAISE;
END;
/

--przykładowe użycie procedury
--zmiana ilości biletów rezerwacji o id 121 na 2
declare
```

```
P_RESERVATION_ID INT := 121;
P_NO_TICKETS      INT := 2;
begin
    BD_420909.P_MODIFY_RESERVATION_6A(
        P_RESERVATION_ID => P_RESERVATION_ID,
        P_NO_TICKETS => P_NO_TICKETS
    );
end;

--p_modify_max_no_places_6a
create PROCEDURE p_modify_max_no_places_6a(
    p_trip_id      IN INT,
    p_max_no_places IN INT
) AS
    v_reserved INT;
BEGIN
    IF NOT f_trip_exists(p_trip_id) THEN
        RAISE_APPLICATION_ERROR(-20010, 'Wycieczka nie istnieje.');
```

END IF;

```
    SELECT NVL(SUM(no_tickets),0)
        INTO v_reserved
        FROM reservation
        WHERE trip_id = p_trip_id AND status IN ('N','P');
```

IF p_max_no_places < v_reserved THEN

```
    RAISE_APPLICATION_ERROR(-20005, 'Nowa maksymalna liczba miejsc jest
mniej niż liczba zarezerwowanych.');
```

END IF;

```
    UPDATE trip
        SET max_no_places = p_max_no_places,
            no_available_places = p_max_no_places - v_reserved
        WHERE trip_id = p_trip_id;
```

EXCEPTION

```
    WHEN OTHERS THEN
        ROLLBACK;
        RAISE;
```

END;

/

--przykładowe użycie procedury

--zmiana ilości maksymalnych miejsc na wycieczkę o id 1 na 201

```
declare
    P_TRIP_ID      INT := 1;
    P_MAX_NO_PLACES INT := 201;
begin
    BD_420909.P_MODIFY_MAX_NO_PLACES_6A(
        P_TRIP_ID => P_TRIP_ID,
        P_MAX_NO_PLACES => P_MAX_NO_PLACES
    );
end;
```

Zadanie 6b - triggerery

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

```
--trigger TRG_UPDATE_NO_AVAILABLE_PLACES_INS_6B
--zmniejsza liczbę dostępnych miejsc w tabeli trip o liczbę biletów z nowej
rezerwacji
create trigger TRG_UPDATE_NO_AVAILABLE_PLACES_INS_6B
  after insert
  on RESERVATION
  for each row
BEGIN
  UPDATE trip
  SET no_available_places = no_available_places - :NEW.no_tickets
  WHERE trip_id = :NEW.trip_id;
END;
/

--trigger TRG_UPDATE_NO_AVAILABLE_PLACES_UPD_6B
--sprawdza zmiany statusu rezerwacji i liczby miejsc
create trigger TRG_UPDATE_NO_AVAILABLE_PLACES_UPD_6B
  before update of STATUS,NO_TICKETS
  on RESERVATION
  for each row
BEGIN
  IF :OLD.status IN ('N','P') THEN
    UPDATE trip
      SET no_available_places = no_available_places + :OLD.no_tickets
      WHERE trip_id = :OLD.trip_id;
  END IF;

  IF :NEW.status IN ('N','P') THEN
```

```
        UPDATE trip
            SET no_available_places = no_available_places - :NEW.no_tickets
            WHERE trip_id = :NEW.trip_id;
    END IF;
END;
/

--nowe wersje procedur
--p_add_reservation_6b
create PROCEDURE p_add_reservation_6b(
    p_trip_id    IN INT,
    p_person_id  IN INT,
    p_no_tickets IN INT
) AS
BEGIN
    IF NOT f_trip_exists(p_trip_id) THEN
        RAISE_APPLICATION_ERROR(-20010, 'Wycieczka nie istnieje.');
```

END IF;

IF NOT f_person_exists(p_person_id) THEN

RAISE_APPLICATION_ERROR(-20011, 'Osoba nie istnieje.');

END IF;

INSERT INTO reservation(trip_id, person_id, status, no_tickets)

VALUES (p_trip_id, p_person_id, 'N', p_no_tickets);

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

RAISE;

END;

/

--przykładowe użycie procedury

--dodanie rezerwacji na wycieczkę o id 1 dla osoby o id 5 z liczbą biletów równą 1

declare

P_TRIP_ID INT := 1;

P_PERSON_ID INT := 5;

P_NO_TICKETS INT := 1;

begin

BD_420909.P_ADD_RESERVATION_6B(

P_TRIP_ID => P_TRIP_ID,

P_PERSON_ID => P_PERSON_ID,

P_NO_TICKETS => P_NO_TICKETS

);

end;

--p_modify_reservation_status_6b

create PROCEDURE p_modify_reservation_status_6b(

p_reservation_id IN INT,

p_status IN VARCHAR2

```
) AS
BEGIN
    IF NOT f_reservation_exists(p_reservation_id) THEN
        RAISE_APPLICATION_ERROR(-20012, 'Rezerwacja nie istnieje.');
```

END IF;

UPDATE reservation

SET status = p_status

WHERE reservation_id = p_reservation_id;

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

RAISE;

END;

/

--przykładowe użycie procedury

--zmiana statusu rezerwacji o numerze 121 na 'P'

declare

P_RESERVATION_ID INT := 121;

P_STATUS VARCHAR2(4000) := 'P';

begin

BD_420909.P_MODIFY_RESERVATION_STATUS_6B(

P_RESERVATION_ID => P_RESERVATION_ID,

P_STATUS => P_STATUS

);

end;

--p_modify_reservation_6b

create PROCEDURE p_modify_reservation_6b(

p_reservation_id IN INT,

p_no_tickets IN INT

) AS

BEGIN

IF NOT f_reservation_exists(p_reservation_id) THEN

RAISE_APPLICATION_ERROR(-20012, 'Rezerwacja nie istnieje.');

END IF;

UPDATE reservation

SET no_tickets = p_no_tickets

WHERE reservation_id = p_reservation_id;

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

RAISE;

END;

/

--przykładowe użycie procedury

--zmiana ilości biletów rezerwacji o id 121 na 2

```
declare
  P_RESERVATION_ID INT := 121;
  P_NO_TICKETS      INT := 2;
begin
  BD_420909.P_MODIFY_RESERVATION_6B(
    P_RESERVATION_ID => P_RESERVATION_ID,
    P_NO_TICKETS => P_NO_TICKETS
  );
end;
```

Zadanie 7 - podsumowanie

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

Oracle PL/SQL umożliwia rozbudowaną obsługę bloków proceduralnych z deklaracją zmiennych, obsługą wyjątków (EXCEPTION) i możliwością definiowania procedur, funkcji oraz pakietów. MS SQL Server T-SQL również umożliwia programowanie proceduralne, ale składnia różni się od PL/SQL (np. deklaracja zmiennych przy użyciu **DECLARE**, obsługa błędów za pomocą TRY...CATCH).

Transakcje w **Oracle** PL są zarządzane jawnie – każde polecenie **COMMIT** lub **ROLLBACK** dotyczy całej sesji. Natomiast w MS Sqlserver transakcje rozpoczynamy za pomocą **BEGIN TRANSACTION**, a następnie zatwierdzamy lub cofamy za pomocą **COMMIT TRANSACTION/ROLLBACK TRANSACTION**.

W **Oracle** PL/SQL obsługa **triggerów** jest bardzo elastyczna – możemy definiować triggerzy na poziomie wiersza lub tabeli, na różne operacje (**INSERT**, **UPDATE**, **DELETE**). **Trigger** w MS Sqlserver T-SQL mogą być definiowane na poziomie tabeli i często operują na zestawach danych, co wymaga innego podejścia.

Różnice w obsłudze błędów i mechanizmach transakcyjnych wpływają na styl programowania – w **Oracle** częściej stosuje się blokowe podejście, podczas gdy w MS Sqlserver dużo operacji odbywa się na zestawach danych.