Oracle PL/Sql

widoki, funkcje, procedury, triggery ćwiczenie

Imiona i nazwiska autorów : Dawid Żak, Szymon Migas

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 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);

insert into trip(trip_name, country, trip_date, max_no_places)
values ('Wakacje w Rzymie', 'Włochy', to_date('2022-07-15', 'YYYY-MM-DD'),
5);

insert into trip(trip_name, country, trip_date, max_no_places)
values ('Zwiedzanie Londynu', 'Wielka Brytania', to_date('2023-03-10',
'YYYY-MM-DD'), 10);

insert into trip(trip_name, country, trip_date, max_no_places)
values ('Safari w Kenii', 'Kenia', to_date('2025-08-20', 'YYYY-MM-DD'),
15);

insert into trip(trip_name, country, trip_date, max_no_places)
values ('Podróż do Tokio', 'Japonia', to_date('2025-12-05', 'YYYY-MM-DD'),
20);

insert into trip(trip_name, country, trip_date, max_no_places)
values ('Rejs po Karaibach', 'Karaiby', to_date('2026-01-15', 'YYYY-MM-
DD'), 25);

insert into trip(trip_name, country, trip_date, max_no_places)
values ('Zwiedzanie Aten', 'Grecja', to_date('2026-04-10', 'YYYY-MM-DD'),
120);

-- 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');

insert into person(firstname, lastname)
values ('Anna', 'Kowalska');

insert into person(firstname, lastname)
values ('Piotr', 'Zieliński');

insert into person(firstname, lastname)
values ('Maria', 'Wiśniewska');

insert into person(firstname, lastname)
values ('Tomasz', 'Lewandowski');

insert into person(firstname, lastname)
values ('Katarzyna', 'Nowicka');

insert into person(firstname, lastname)
values ('Michał', 'Szymański');

-- 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');

-- trip 4

-- trip 5
insert into reservation(trip_id, person_id, status, no_tickets)
values (5, 3, 'P', 2);

insert into reservation(trip_id, person_id, status, no_tickets)
values (5, 6, 'N', 1);
```

```
-- trip 6
insert into reservation(trip_id, person_id, status, no_tickets)
values (6, 2, 'P', 3);

insert into reservation(trip_id, person_id, status, no_tickets)
values (6, 4, 'C', 1);

-- trip 7
insert into reservation(trip_id, person_id, status, no_tickets)
values (7, 5, 'P', 4);

insert into reservation(trip_id, person_id, status, no_tickets)
values (7, 1, 'N', 2);

-- trip 8
insert into reservation(trip_id, person_id, status, no_tickets)
values (8, 7, 'P', 5);

insert into reservation(trip_id, person_id, status, no_tickets)
values (8, 8, 'N', 3);

-- trip 9
insert into reservation(trip_id, person_id, status, no_tickets)
values (9, 9, 'P', 6);

insert into reservation(trip_id, person_id, status, no_tickets)
values (9, 10, 'C', 2);

-- trip 10
insert into reservation(trip_id, person_id, status, no_tickets)
values (10, 1, 'P', 1);

insert into reservation(trip_id, person_id, status, no_tickets)
values (10, 2, 'N', 4);
```

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 dodac pole
  - no_tickets

Należy zmodyfikować zestaw danych testowych

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

Skomentuj dzialanie 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=311899 w szczególności dokument: `1_ora_modyf.pdf`

```
ALTER TABLE reservation
ADD no_tickets INT DEFAULT 1;

ALTER TABLE log
ADD no_tickets INT;
```

Transakcje w Oracle PL/SQL służą do bezpieczniejszego przeprowadzania operacji na bazie danych. Dają nam możliwość cofnięcia ostatnich zmian, które mogły spowodować różnego rodzaju problemy (za pomocą polecenia rollback). W przypadku wystąpienia błędów w transakcji polecenie COMMIT nie zostanie wykonane automatycznie, będziemy mogli poprawić błąd lub cofnąć wcześniej wykonane polecenia za pomocą ROLLBACK.

```
SET TRANSACTION READ WRITE -- domyślne ustawienia w Oracle Sql;
-- Dodanie osoby do bazy
    INSERT INTO person (firstname, lastname)
    VALUES ('Jan', 'Nowak');

-- Dodanie rezerwacji do bazy
    INSERT INTO reservation (trip_id, person_id, status, no_tickets)
    VALUES (1, 1, 'P', 2);
COMMIT
```

Oracle vs. MS SQL

- W MS SQL transakcje muszą być rozpoczynane jawnie poprzez BEGIN TRANSACTION, natomiast w Oracle transakcja jest rozpoczynana przy pierwszym poleceniu UPDATE, INSERT, DELETE (w trybie write)
- Obsługa błędów w Oracle odbywa się za pomocą BEGIN...EXCEPTION, gdzie w MS SQL stosowało się składnie bardziej zbliżoną do TRY/CATCH

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

Widok `vw_reservation`

```
CREATE
OR REPLACE VIEW vw_reservation AS
SELECT
    r.reservation_id,
    t.country,
    t.trip_date,
    t.trip_name,
    p.firstname,
    p.lastname,
    r.status,
    r.trip_id,
    r.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;
```

Widok `vw_trip`

```
CREATE
OR REPLACE VIEW vw_trip AS
SELECT
    t.trip_id,
    t.country,
    t.trip_date,
    t.trip_name,
    t.max_no_places,
    t.max_no_places - NVL(SUM(r.no_tickets), 0) AS no_available_places
FROM
    trip t
LEFT JOIN
    reservation r ON t.trip_id = r.trip_id AND r.status != 'C'
GROUP BY
    t.trip_id, t.country, t.trip_date, t.trip_name, t.max_no_places;
```

Widok `vw_available_trip`

```
CREATE OR REPLACE VIEW vw_available_trip AS
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;
```

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?

Proponowany zestaw funkcji można rozbudować wedle uznania/potrzeb

- np. można dodać nowe/pomocnicze funkcje/procedury

Zadanie 2 - rozwiązanie

Zaczęliśmy od zdefiniowania typu danych, które będą zwracane przez dwie pierwsze funkcje, mają to być dane podobne do zestawu z widoku `vw_reservation`

```
CREATE OR REPLACE TYPE reservation_info AS OBJECT
(
    RESERVATION_ID NUMBER,
    COUNTRY VARCHAR2(50),
    TRIP_DATE DATE,
    TRIP_NAME VARCHAR2(100),
    FIRSTNAME VARCHAR2(50),
    LASTNAME VARCHAR2(50),
    STATUS CHAR,
    TRIP_ID NUMBER,
    PERSON_ID NUMBER,
    NO_TICKETS NUMBER
);

-- Typ danych dla wyjścia funkcji
CREATE OR REPLACE TYPE reservation_info_table IS TABLE OF reservation_info;
```

Podobnie dla trzeciej funkcji i widoku `vw_available_trips`

```
create type available_trips_info as object
(
    TRIP_ID NUMBER,
    COUNTRY VARCHAR2(50),
    TRIP_DATE DATE,
    TRIP_NAME VARCHAR2(100),
    MAX_NO_PLACES NUMBER,
    NO_AVAILABLE_PLACES NUMBER
);

-- Typ danych dla wyjścia funkcji
create type available_trips_info_table is table of available_trips_info;
```

Funkcja f_trip_participants

```
create or replace function f_trip_participants(trip_id varchar)
    return reservation_info_table
as
    result reservation_info_table;
begin
    select RESERVATION_INFO(vw.RESERVATION_ID, vw.COUNTRY, vw.TRIP_DATE,
vw.TRIP_NAME, vw.FIRSTNAME, vw.LASTNAME,
                            vw.STATUS, vw.TRIP_ID, vw.PERSON_ID,
vw.NO_TICKETS) bulk collect
    into result
    from vw_reservation vw
    where vw.trip_id = f_trip_participants.trip_id;
    return result;
end;

-- Przykład wywołania funkcji
select * from f_trip_participants(1)
```

Funkcja f_person_reservations

```
create or replace function f_person_reservation(person_id NUMBER)
    return RESERVATION_INFO_TABLE
as
    result RESERVATION_INFO_TABLE;
begin
    select RESERVATION_INFO(vw.RESERVATION_ID, vw.COUNTRY, vw.TRIP_DATE,
vw.TRIP_NAME, vw.FIRSTNAME, vw.LASTNAME,
                            vw.STATUS, vw.TRIP_ID, vw.PERSON_ID,
vw.NO_TICKETS) bulk collect
    into result
    from VW_RESERVATION vw
    where vw.PERSON_ID = f_person_reservation.person_id;
```

```
    return result;
end;

-- Przykład wywołania funkcji
select * from f_person_reservation(1);
```

Funkcja `f_available_trips_to`

```
create or replace function f_available_trips_to(country varchar2, date_from
DATE, date_to DATE)
    return AVAILABLE_TRIPS_INFO_TABLE
as
    result AVAILABLE_TRIPS_INFO_TABLE;
begin
    select AVAILABLE_TRIPS_INFO(vw.TRIP_ID, vw.COUNTRY, vw.TRIP_DATE,
vw.TRIP_NAME, vw.MAX_NO_PLACES, vw.NO_AVAILABLE_PLACES) bulk collect
    into result
    from VW_AVAILABLE_TRIP vw
    where
        vw.COUNTRY = f_available_trips_to.country
    and
        vw.TRIP_DATE between date_from and date_to;
    return result;
end;

-- Przykład wywołania funkcji
select * from F_AVAILABLE_TRIPS_TO('Polska', '2024-01-01', '2025-11-12')
```

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 sa 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

Procedura `p_add_reservation`

```
create procedure p_add_reservation(trip_id int, person_id int, no_tickets
int)
as
    v_trip_date        DATE;
    v_available_places NUMBER;
    v_reservation_id   NUMBER;
    v_count            NUMBER;
begin

    SELECT COUNT(*) INTO v_count
    FROM trip
    WHERE trip_id = p_add_reservation.trip_id;

    IF v_count = 0 THEN
        RETURN;
    END IF;
```

```
    SELECT COUNT(*) INTO v_count
    FROM person
    WHERE person_id = p_add_reservation.person_id;

    IF v_count = 0 THEN
        RETURN;
    END IF;

    SELECT trip_date, no_available_places
    INTO v_trip_date, v_available_places
    FROM vw_trip
    WHERE trip_id = p_add_reservation.trip_id;

    IF v_trip_date < SYSDATE THEN
        RETURN;
    END IF;

    IF v_available_places < no_tickets THEN
        RETURN;
    END IF;

    INSERT INTO RESERVATION (trip_id, person_id, status, no_tickets)
    VALUES (trip_id, person_id, 'N', no_tickets)
    RETURNING reservation_id INTO v_reservation_id;

    INSERT INTO LOG (reservation_id, log_date, status, no_tickets)
    VALUES (v_reservation_id, SYSDATE, 'N', no_tickets);

    COMMIT;
end;
```

Procedura p_modify_reservation_status

```
create or replace procedure p_modify_reservation_status(p_reservation_id
INT, p_status CHAR)
as
    v_current_status CHAR(1);
    v_trip_id NUMBER;
    v_no_tickets NUMBER;
    v_trip_date DATE;
    v_available_places NUMBER;
    v_count NUMBER;
begin
    SELECT COUNT(*) INTO v_count
    FROM reservation
    WHERE reservation_id = p_reservation_id;

    IF v_count = 0 THEN
        RETURN;
```

```
    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;


    IF v_current_status = 'C' AND (p_status = 'P' OR p_status = 'N') THEN

        SELECT COUNT(*) INTO v_count
        FROM trip
        WHERE trip_id = v_trip_id;

        IF v_count = 0 THEN
            RETURN;
        END IF;


        SELECT trip_date, no_available_places
        INTO v_trip_date, v_available_places
        FROM vw_trip
        WHERE trip_id = v_trip_id;


        IF v_trip_date < SYSDATE THEN
            RETURN;
        END IF;


        IF v_available_places < v_no_tickets THEN
            RETURN;
        END IF;
    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, v_no_tickets);

    COMMIT;
end;
```

Procedura p_modify_reservation

```
create or replace procedure p_modify_reservation(p_reservation_id INT,
p_no_tickets INT)
as
    v_current_tickets NUMBER;
    v_status CHAR(1);
    v_trip_id NUMBER;
    v_trip_date DATE;
    v_available_places NUMBER;
    v_count NUMBER;
    v_additional_tickets NUMBER;
begin

    SELECT COUNT(*) INTO v_count
    FROM reservation
    WHERE reservation_id = p_reservation_id;

    IF v_count = 0 THEN
        RETURN;
    END IF;


    SELECT no_tickets, status, trip_id
    INTO v_current_tickets, v_status, v_trip_id
    FROM reservation
    WHERE reservation_id = p_reservation_id;


    IF v_current_tickets = p_no_tickets THEN
        RETURN;
    END IF;


    IF v_status != 'C' AND p_no_tickets > v_current_tickets THEN

        v_additional_tickets := p_no_tickets - v_current_tickets;


        SELECT COUNT(*) INTO v_count
        FROM trip
        WHERE trip_id = v_trip_id;

        IF v_count = 0 THEN
            RETURN;
        END IF;


        SELECT trip_date, no_available_places
        INTO v_trip_date, v_available_places
        FROM vw_trip
        WHERE trip_id = v_trip_id;
```

```
        IF v_trip_date < SYSDATE THEN
            RETURN;
        END IF;


        IF v_available_places < v_additional_tickets THEN
            RETURN;
        END IF;
    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, v_status, p_no_tickets);

    COMMIT;
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_current_max_places NUMBER;
    v_reserved_places NUMBER;
    v_count NUMBER;
begin

    SELECT COUNT(*) INTO v_count
    FROM vw_trip
    WHERE trip_id = p_trip_id;

    IF v_count = 0 THEN
        RETURN;
    END IF;


    SELECT max_no_places, (max_no_places - no_available_places) INTO
v_current_max_places, v_reserved_places
    FROM vw_trip
    WHERE trip_id = p_trip_id;


    IF v_current_max_places = p_max_no_places THEN
```

```
        RETURN;
    END IF;


    IF p_max_no_places < v_reserved_places THEN
        RETURN;
    END IF;


    UPDATE trip
    SET max_no_places = p_max_no_places
    WHERE trip_id = p_trip_id;

    COMMIT;
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 trigerów

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

W celu modyfikacji procedur wystarczy usunąć część definicji procedury z poleceniem `INSERT` na tabeli `LOG`.

Trigger obsługujący dodanie nowej rezerwacji: `tr_reservation_insert_log`

Po wstawieniu danych do tabeli `reservation` dodaje wpis do tabeli `log` informujący o zmianach.

```
CREATE OR REPLACE TRIGGER tr_reservation_insert_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;
```

Zmodyfikowana procedura `p_add_reservation`

```
create procedure p_add_reservation_4(trip_id int, person_id int, no_tickets
int)
as
    v_trip_date DATE;
    v_available_places NUMBER;
    v_reservation_id NUMBER;
v_count            NUMBER;
begin

    SELECT COUNT(*) INTO v_count
    FROM trip
    WHERE trip_id = p_add_reservation_4.trip_id;

    IF v_count = 0 THEN
        RETURN;
    END IF;

    SELECT COUNT(*) INTO v_count
    FROM person
    WHERE person_id = p_add_reservation_4.person_id;

    IF v_count = 0 THEN
        RETURN;
    END IF;
    SELECT trip_date, no_available_places
    INTO v_trip_date, v_available_places
    FROM vw_trip
    WHERE trip_id = p_add_reservation_4.trip_id;

    IF v_trip_date < SYSDATE THEN
        RETURN;
    END IF;
```

```
    IF v_available_places < no_tickets THEN
        RETURN;
    END IF;

    INSERT INTO RESERVATION (trip_id, person_id, status, no_tickets)
    VALUES (trip_id, person_id, 'N', no_tickets)
    RETURNING reservation_id INTO v_reservation_id;
end;
```

Trigger obsługujący zmianę statusu rezerwacji `tr_reservation_status_update_log`

Po aktualizacji statusu rezerwacji dodawany jest wpis do tabeli `log` informujący o zmianach

```
CREATE OR REPLACE TRIGGER tr_reservation_status_update_log
AFTER UPDATE OF status ON reservation
FOR EACH ROW
WHEN (OLD.status != NEW.status)
BEGIN
    INSERT INTO log (reservation_id, log_date, status, no_tickets)
    VALUES (:NEW.reservation_id, SYSDATE, :NEW.status, :NEW.no_tickets);
END;
```

Zmodyfikowana procedura `p_modify_reservation_status`

```
create or replace procedure p_modify_reservation_status_4(p_reservation_id
INT, p_status CHAR)
as
    v_current_status CHAR(1);
    v_trip_id NUMBER;
    v_no_tickets NUMBER;
    v_trip_date DATE;
    v_available_places NUMBER;
    v_count NUMBER;
begin
    SELECT COUNT(*) INTO v_count
    FROM reservation
    WHERE reservation_id = p_reservation_id;

    IF v_count = 0 THEN
        RETURN;
    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;


    IF v_current_status = 'C' AND (p_status = 'P' OR p_status = 'N') THEN

        SELECT COUNT(*) INTO v_count
        FROM trip
        WHERE trip_id = v_trip_id;

        IF v_count = 0 THEN
            RETURN;
        END IF;


        SELECT trip_date, no_available_places
        INTO v_trip_date, v_available_places
        FROM vw_trip
        WHERE trip_id = v_trip_id;


        IF v_trip_date < SYSDATE THEN
            RETURN;
        END IF;


        IF v_available_places < v_no_tickets THEN
            RETURN;
        END IF;
    END IF;


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

Trigger obsługujący zmianę liczby zarezerwowanych/kupionych biletów `tr_reservation_tickets_update_log`

Po aktualizacji liczby zakupionych/zarezerwowanych biletów jest dodawany odpowiedni rekord w tabeli `log`

```
CREATE OR REPLACE TRIGGER tr_reservation_tickets_update_log
AFTER UPDATE OF no_tickets ON reservation
FOR EACH ROW
WHEN (OLD.no_tickets != NEW.no_tickets)
BEGIN
```

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

Zmodyfikowana procedura `p_modify_reservation`

```
create or replace procedure p_modify_reservation_4(p_reservation_id INT,
p_no_tickets INT)
as
    v_current_tickets NUMBER;
    v_status CHAR(1);
    v_trip_id NUMBER;
    v_trip_date DATE;
    v_available_places NUMBER;
    v_count NUMBER;
    v_additional_tickets NUMBER;
begin

    SELECT COUNT(*) INTO v_count
    FROM reservation
    WHERE reservation_id = p_reservation_id;

    IF v_count = 0 THEN
        RETURN;
    END IF;


    SELECT no_tickets, status, trip_id
    INTO v_current_tickets, v_status, v_trip_id
    FROM reservation
    WHERE reservation_id = p_reservation_id;


    IF v_current_tickets = p_no_tickets THEN
        RETURN;
    END IF;


    IF v_status != 'C' AND p_no_tickets > v_current_tickets THEN

        v_additional_tickets := p_no_tickets - v_current_tickets;


        SELECT COUNT(*) INTO v_count
        FROM trip
        WHERE trip_id = v_trip_id;

        IF v_count = 0 THEN
            RETURN;
        END IF;
```

```
        SELECT trip_date, no_available_places
        INTO v_trip_date, v_available_places
        FROM vw_trip
        WHERE trip_id = v_trip_id;


        IF v_trip_date < SYSDATE THEN
            RETURN;
        END IF;


        IF v_available_places < v_additional_tickets THEN
            RETURN;
        END IF;
    END IF;


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

Trigger zabraniający usunięcia rezerwacji

W celu archiwizacji trigger `tr_prevent_reservation_delete` zabrania usuwania rezerwacji

```
CREATE OR REPLACE TRIGGER tr_prevent_reservation_delete
BEFORE DELETE ON reservation
FOR EACH ROW
BEGIN
    RAISE_APPLICATION_ERROR(-20001, 'Deletion of reservations is not
allowed');
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 trigerów

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_5`

Zadanie 5 - rozwiązanie

Trigger obsługujący dodawanie rezerwacji

Trigger `tr_insert_reservation` zapewnia, że rezerwacja spełnia wszystkie warunki aby mogła zostać wstawiona do bazy.

```
create or replace trigger TR_INSERT_RESERVATION
    before insert
    on RESERVATION
    for each row
declare
    v_count int;
    v_trip_date date;
    v_available_places int;
begin
    select count(*)
    into v_count
    from trip t
    where t.trip_id = :new.trip_id;

    if v_count = 0 then
        RAISE_APPLICATION_ERROR(-20010, 'Trip does not exist!');
    end if;

    select count(*)
    into v_count
    from person p
    where p.person_id = :new.person_id;

    if v_count = 0 then
        RAISE_APPLICATION_ERROR(-20020, 'Person does not exist!');
    end if;

    select trip_date, no_available_places
```

```
    into v_trip_date, v_available_places
    from vw_trip
    where trip_id = :new.trip_id;

    if v_trip_date < SYSDATE then
        RAISE_APPLICATION_ERROR(-20030, 'Trip date from the past!');
    end if;

    IF v_available_places < :new.no_tickets then
        RAISE_APPLICATION_ERROR(-20040, 'No available tickets!');
    end if;

end;
```

Trigger obsługujący zmianę statusu rezerwacji oraz zmianę liczby biletów

Trigger `tr_reservation_update` odpowiada jednocześnie za zmianę statusu oraz liczby biletów podczas modyfikacji tabeli `RESERVATION`

Zakładamy, że możliwa jest zmiana liczby biletów niezależnie od statusu, jednakże niemożliwa jest modyfikacja jeżeli wycieczka już się odbyła

```
create trigger TR_RESERVATION_UPDATE
    before update
    on RESERVATION
    for each row
declare
    v_count            int;
    v_trip_date        date;
    v_available_places int;
begin
    select count(*)
    into v_count
    from reservation r
    where r.reservation_id = :new.reservation_id;

    if v_count = 0 then
        RAISE_APPLICATION_ERROR(-20020, 'Reservation does not exist!');
    end if;

    select trip_date, no_available_places
    into v_trip_date, v_available_places
    from vw_trip
    where trip_id = :new.trip_id;


    if v_trip_date < SYSDATE then
        RAISE_APPLICATION_ERROR(-20030, 'Trip date from the past!');
    end if;

    IF :old.status = 'C' AND (:new.status = 'P') THEN
```

```
        SELECT COUNT(*)
        INTO v_count
        FROM trip
        WHERE trip_id = :old.trip_id;

        IF v_count = 0 THEN
            RETURN;
        END IF;

        IF v_available_places < :new.no_tickets THEN
            RAISE_APPLICATION_ERROR(-20040, 'No available tickets!');
        END IF;
    END IF;

    if v_available_places < :new.no_tickets - :old.no_tickets then
        RAISE_APPLICATION_ERROR(-20040, 'No available tickets!');
    end if;

end;
```

Zmodyfikowane procedury

W każdej procedurze po podniesionym przez trigger błędzie zapewniana jest jego obsługa.

Procedura `p_add_reservation`

```
CREATE OR REPLACE PROCEDURE p_add_reservation_5(trip_id INT, person_id INT,
no_tickets INT)
AS
BEGIN
    BEGIN
        INSERT INTO RESERVATION (trip_id, person_id, status, no_tickets)
        VALUES (trip_id, person_id, 'N', no_tickets);
    EXCEPTION
        WHEN OTHERS THEN
            DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
    END;
END;
```

Procedura `p_modify_reservation_status`

```
    CREATE OR REPLACE PROCEDURE
p_modify_reservation_status_5(p_reservation_id INT, p_status CHAR)
AS
    v_count NUMBER;
    v_current_status CHAR(1);
BEGIN
```

```
    SELECT COUNT(*) INTO v_count
    FROM reservation
    WHERE reservation_id = p_reservation_id;

    IF v_count = 0 THEN
        RETURN;
    END IF;


    SELECT status INTO v_current_status
    FROM reservation
    WHERE reservation_id = p_reservation_id;


    IF v_current_status = p_status THEN
        RETURN;
    END IF;


    BEGIN
        UPDATE reservation
        SET status = p_status
        WHERE reservation_id = p_reservation_id;
    EXCEPTION
        WHEN OTHERS THEN
            DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
    END;
END;
```

Procedura p_modify_reservation

```
CREATE OR REPLACE PROCEDURE p_modify_reservation_5(p_reservation_id INT,
p_no_tickets INT)
AS
    v_count NUMBER;
    v_current_tickets NUMBER;
BEGIN

    SELECT COUNT(*) INTO v_count
    FROM reservation
    WHERE reservation_id = p_reservation_id;

    IF v_count = 0 THEN
        RETURN;
    END IF;


    SELECT no_tickets INTO v_current_tickets
    FROM reservation
    WHERE reservation_id = p_reservation_id;
```

```
    IF v_current_tickets = p_no_tickets THEN
        RETURN;
    END IF;


    BEGIN
        UPDATE reservation
        SET no_tickets = p_no_tickets
        WHERE reservation_id = p_reservation_id;
    EXCEPTION
        WHEN OTHERS THEN
            DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
    END;
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

Po zmianie struktury tabeli przystąpiliśmy do uzupełniania wartości pola `no_available_places` za pomocą polecenia

```
UPDATE trip t
SET no_available_places = (
    SELECT t.max_no_places - NVL(SUM(r.no_tickets), 0)
    FROM reservation r
    WHERE r.trip_id = t.trip_id
    AND r.status != 'C'
    GROUP BY r.trip_id
);
```

Aby obsłużyć przypadek, gdy w tabeli `reservation` nie istnieje żaden rekord dla danej wycieczki wykonaliśmy kolejne polecenie:

```
UPDATE trip t
SET no_available_places = max_no_places
WHERE no_available_places IS NULL;
```

Nowe widoki

Widok `vw_available_trip`

```
create or replace view vw_available_trip_6 as
select trip_id,
       country,
       trip_date,
       trip_name,
       max_no_places,
       no_available_places
from trip
where trip_date > SYSDATE
  and no_available_places > 0
```

Widok `vw_trip`

```
create or replace view vw_trip_6 as
select t.trip_id,
       t.country,
       t.trip_date,
       t.trip_name,
```

```
        t.max_no_places,
        t.no_available_places
from trip t
```

Widok `vw_reservation` pozostaje bez zmian, ponieważ nie korzysta on z pola `no_available_places`

Zmiana funkcji

Każda funkcja korzystająca ze starych widoków, została zmieniona tak, aby korzystała z nowych widoków *(z końcówką '_6')*.

Zmiana triggerów

Podobnie jak w przypadku funkcji, zmiana obsługi polegała na zmienie metody pobierania informacji o wolnych miejscach

Uwaga

Wszystkie zmienione triggery i metody są umieszczone w pliku `polecenia.sql`

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` służy do dodawania nowej rezerwacji korzystając z pola `no_available_places` w tabeli `trip`, jeżeli rezerwacja nie może zostać dodana wyłapie to trigger

```
CREATE OR REPLACE PROCEDURE p_add_reservation_6a(trip_id INT, person_id
INT, no_tickets INT)
AS
BEGIN
    BEGIN
        INSERT INTO RESERVATION (trip_id, person_id, status, no_tickets)
        VALUES (trip_id, person_id, 'N', no_tickets);

        UPDATE trip
        SET no_available_places = no_available_places - no_tickets
        WHERE trip_id = p_add_reservation_6a.trip_id;


        COMMIT;
    EXCEPTION
        WHEN OTHERS THEN
            ROLLBACK;
            DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
    END;
END;
```

Procedura `p_modify_reservation_6a` służy do zmiany ilości zarezerwowanych/kupionych biletów, uwzględniając pole `no_available_places` w tabeli `trip`

```
CREATE OR REPLACE PROCEDURE p_modify_reservation_6a(p_reservation_id INT,
p_no_tickets INT)
AS
    v_count NUMBER;
    v_current_tickets NUMBER;
    v_status CHAR(1);
    v_trip_id NUMBER;
    v_old_count NUMBER := 0;
    v_new_count NUMBER := 0;
BEGIN

    SELECT COUNT(*) INTO v_count
    FROM reservation
    WHERE reservation_id = p_reservation_id;

    IF v_count = 0 THEN
        RETURN;
    END IF;

    SELECT trip_id, no_tickets, status
    INTO v_trip_id, v_current_tickets, v_status
    FROM reservation
    WHERE reservation_id = p_reservation_id;
```

```
    IF v_current_tickets = p_no_tickets THEN
        RETURN;
    END IF;


    IF v_status != 'C' THEN
        v_old_count := v_current_tickets;
        v_new_count := p_no_tickets;
    END IF;

    BEGIN

        UPDATE reservation
        SET no_tickets = p_no_tickets
        WHERE reservation_id = p_reservation_id;


        IF v_status != 'C' THEN
            UPDATE trip
            SET no_available_places = no_available_places + (v_old_count -
v_new_count)
            WHERE trip_id = v_trip_id;
        END IF;

    EXCEPTION
        WHEN OTHERS THEN
            ROLLBACK;
            DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
    END;
END;
```

Trigger `p_modify_reservation_status_6a` służący do modyfikacji statusu rezerwacji

```
CREATE OR REPLACE PROCEDURE p_modify_reservation_status_6a(p_reservation_id
INT, p_status CHAR)
AS
    v_count NUMBER;
    v_current_status CHAR(1);
    v_trip_id NUMBER;
    v_no_tickets NUMBER;
    v_old_count NUMBER := 0;
    v_new_count NUMBER := 0;
BEGIN
    SELECT COUNT(*) INTO v_count
    FROM reservation
    WHERE reservation_id = p_reservation_id;

    IF v_count = 0 THEN
        RETURN;
    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;


    IF v_current_status != 'C' AND p_status = 'C' THEN

        v_old_count := v_no_tickets;
    ELSIF v_current_status = 'C' AND p_status != 'C' THEN

        v_new_count := v_no_tickets;
    END IF;

    BEGIN

        UPDATE reservation
        SET status = p_status
        WHERE reservation_id = p_reservation_id;


        IF v_old_count > 0 OR v_new_count > 0 THEN
            UPDATE trip
            SET no_available_places = no_available_places + (v_old_count -
v_new_count)
            WHERE trip_id = v_trip_id;
        END IF;


    EXCEPTION
        WHEN OTHERS THEN
            ROLLBACK;
            DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
    END;
END;
```

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

Do obsługi dodawania, oraz modyfikacji rezerwacji nową metodą stworzyliśmy nowy trigger `tr_reservation_update_places`.

```
create or replace trigger TR_RESERVATION_UPDATE_PLACES
    after insert or update
    on RESERVATION
    for each row
DECLARE
    v_old_count NUMBER := 0;
    v_new_count NUMBER := 0;
    v_trip_id NUMBER;
BEGIN
    IF INSERTING THEN
        v_trip_id := :NEW.trip_id;
        IF :NEW.status != 'C' THEN
            v_new_count := :NEW.no_tickets;
        END IF;
    ELSIF UPDATING THEN
        v_trip_id := :NEW.trip_id;
        -- Cofanie rezerwacji
        IF :OLD.status != 'C' AND :NEW.status = 'C' THEN
            v_old_count := :OLD.no_tickets;
        -- Przywracanie rezerwacji
        ELSIF :OLD.status = 'C' AND :NEW.status != 'C' THEN
            v_new_count := :NEW.no_tickets;
        -- Zmiana między 'N' a 'P'
        ELSIF :OLD.status != 'C' AND :NEW.status != 'C' THEN
            v_old_count := :OLD.no_tickets;
            v_new_count := :NEW.no_tickets;
        END IF;
    END IF;

    IF v_old_count != v_new_count THEN
        UPDATE trip
        SET no_available_places = no_available_places + (v_old_count -
v_new_count)
        WHERE trip_id = v_trip_id;
    END IF;
END;
```

Trigger `tr_trip_update` służący do aktualizacji pozostałych miejsc, po zwiększeniu limitu na daną wycieczkę

```
CREATE OR REPLACE TRIGGER tr_trip_update
BEFORE UPDATE OF max_no_places ON trip
FOR EACH ROW
DECLARE
    v_reserved_places NUMBER;
BEGIN
    SELECT NVL(SUM(r.no_tickets), 0) INTO v_reserved_places
    FROM reservation r
    WHERE r.trip_id = :NEW.trip_id
    AND r.status != 'C';

    IF :NEW.max_no_places < v_reserved_places THEN
        RAISE_APPLICATION_ERROR(-20060, 'Cannot reduce max places below
already reserved count');
    END IF;

    :NEW.no_available_places := :NEW.max_no_places - v_reserved_places;
END;
```

Zadanie 7 - podsumowanie

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

Największą różnicą dla nas było zastosowanie składni *commit* i *rollback* podczas wykonywania transakcji z bazą danych. Pozwoliły one na sprawdzenie, czy wykonywane przez nas operacje nie miałyby negatywnych konsekwencji dla danych przechowywanych w bazie.

Większość pozostałych konceptów nie różniła się zbytnio od poznanych przez nas wcześniej. Jedną z nowości był blok `begin/end`, który nie służy jedynie do grupowania transakcji, lecz także odpowiada za łapanie błędów, przy pomocy `exception`.

Język Oracle PL/SQL jest bardziej werbalny niż znany nam T-SQL, znaczna liczba operacji wymaga większej ilości kodu.

Obsługa błędów za pomocą `try/catch` w MS SQL była dla nas bardziej intuicyjna, ponieważ powielała ona te same praktyki, które poznaliśmy w innych językach programowania.