Oracle PL/Sql

widoki, funkcje, procedury, triggery
ćwiczenie

Imiona i nazwiska autorów : Kacper Wdowiak, Radosław Szepielak

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.

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

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

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

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

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

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

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

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

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

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

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

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

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

proszę pamiętać o zatwierdzeniu transakcji

Zadanie 0 - modyfikacja danych, transakcje

Należy 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`

*** 1. W poniższym przykładzie nie zostanie dodany żaden wiersz: ***

```
BEGIN
    INSERT INTO reservation(trip_id, person_id, status, no_tickets) VALUES (3, 11, 'ABC', 2);
    INSERT INTO person (firstname, lastname) VALUES ('Anna', 'Kowalczyk');
    COMMIT;
EXCEPTION
    WHEN OTHERS THEN
        ROLLBACK;
        DBMS_OUTPUT.PUT_LINE('Transaction reversed due to: ' || SQLERRM);
END;
```

Otrzymamy na standardowym wyjściu następujący komunikat:

```
BD_420891> BEGIN
               INSERT INTO reservation(trip_id, person_id, status, no_tickets) VALUES (3, 11, 'ABC', 2);
               INSERT INTO person (firstname, lastname) VALUES ('Anna', 'Kowalczyk');
               COMMIT;
           EXCEPTION
               WHEN OTHERS THEN
                   ROLLBACK;
                   DBMS_OUTPUT.PUT_LINE('Transaction reversed due to: ' || SQLERRM);
           END;
[2025-03-17 20:13:29] completed in 19 ms
Transaction reversed due to: ORA-12899: wartość zbyt duża dla kolumny "BD_420891"."RESERVATION"."STATUS" (obecna: 3, maksymalna: 1)
```

Tutaj również nie zostanie dodany wiersz.

```
BEGIN
    INSERT INTO reservation(trip_id, person_id, status, no_tickets) VALUES (1000, 11, 'P', 2);
    COMMIT;
EXCEPTION
    WHEN OTHERS THEN
        ROLLBACK;
        DBMS_OUTPUT.PUT_LINE('Transaction reversed due to: ' || SQLERRM);
        raise;
END;
```

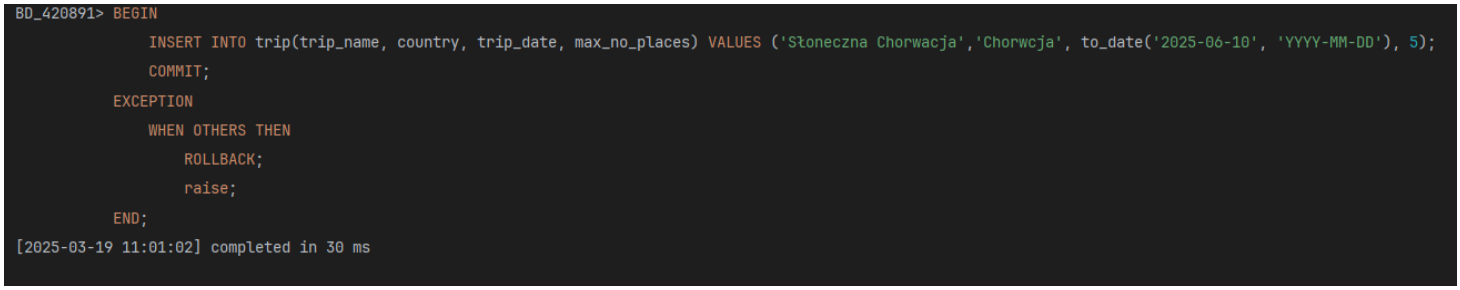
Dla odmiany nie obsługujemy wyjątku przez co otrzymujemy błąd na wyjściu:

```
BD_420891> BEGIN
               INSERT INTO reservation(trip_id, person_id, status, no_tickets) VALUES (1000, 11, 'P', 2);
               COMMIT;
           EXCEPTION
               WHEN OTHERS THEN
                   ROLLBACK;
                   DBMS_OUTPUT.PUT_LINE('Transaction reversed due to: ' || SQLERRM);
                   raise;
           END;
[2025-03-19 10:51:26] [23000][2291]
[2025-03-19 10:51:26]   ORA-02291: naruszono więzy spójności (BD_420891.RESERVATION_FK2) - nie znaleziono klucza nadrzędnego
[2025-03-19 10:51:26]   ORA-06512: przy linia 8
[2025-03-19 10:51:26]   ORA-06512: przy linia 2
[2025-03-19 10:51:26] Position: 0
```

W poniższym przykładzie wiersz zostanie dodany pomyślnie zgodnie z oczekiwaniami.

```
BEGIN
    INSERT INTO trip(trip_name, country, trip_date, max_no_places) VALUES ('Słoneczna Chorwacja','Chorwcja', to_dat
    COMMIT;
EXCEPTION
    WHEN OTHERS THEN
        ROLLBACK;
        raise;
END;
```

Tutaj nie dostajemy wyjątku, ponieważ dane są poprawne - commitujemy zmiany.



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

```
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,
    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;


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 - COALESCE(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;
```

```sql
create or replace view vw_available_trip
as
select t.trip_id,
       t.trip_name,
       t.country,
       t.trip_date,
       t.max_no_places,
       t.max_no_places - COALESCE(sum(r.no_tickets), 0) as number_left
from TRIP t
         left join RESERVATION r on r.trip_id = t.trip_id and r.status != 'C'
where t.trip_date > SYSDATE
group by t.trip_id, t.trip_name, t.country, t.trip_date, t.max_no_places
having t.max_no_places - COALESCE(sum(r.no_tickets), 0) > 0;
```

Zadanie 2 - funkcje

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

Procedury:

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

Funkcje powinny zwracać tabelę/zbiór wynikowy. Należy rozważyć dodanie kontroli parametrów, (np. jeśli parametrem jest `trip_id` to można sprawdzić czy taka

wycieczka istnieje). Podobnie jak w przypadku widoków należy zwrócić uwagę na strukturę kodu

Czy kontrola parametrów w przypadku funkcji ma sens?

- jakie są zalety/wady takiego rozwiązania?

Proponowany zestaw funkcji można rozbudować wedle uznania/potrzeb

- np. można dodać nowe/pomocnicze funkcje/procedury

Zadanie 2 - rozwiązanie

```
CREATE OR REPLACE FUNCTION F_TRIP_PARTICIPANTS(P_TRIP_ID INT)
RETURN SYS_REFCURSOR
AS
    V_CURSOR SYS_REFCURSOR;
BEGIN
    OPEN V_CURSOR FOR
    SELECT RESERVATION_ID, COUNTRY, TRIP_DATE, TRIP_NAME, FIRSTNAME, LASTNAME, STATUS, TRIP_ID, PERSON_ID, NO_TICKE
    FROM VW_RESERVATION
    WHERE TRIP_ID = P_TRIP_ID;
    RETURN V_CURSOR;
END F_TRIP_PARTICIPANTS;


CREATE OR REPLACE FUNCTION F_PERSON_RESERVATIONS(P_PERSON_ID INT)
RETURN SYS_REFCURSOR
AS
    V_CURSOR SYS_REFCURSOR;
BEGIN
    OPEN V_CURSOR FOR
    SELECT RESERVATION_ID, COUNTRY, TRIP_DATE, TRIP_NAME, FIRSTNAME, LASTNAME, STATUS, TRIP_ID, PERSON_ID, NO_TICKE
    FROM VW_RESERVATION
    WHERE PERSON_ID = P_PERSON_ID;
    RETURN V_CURSOR;
END F_PERSON_RESERVATIONS;
```

```
CREATE OR REPLACE FUNCTION f_available_trips_to(p_country VARCHAR2, p_date_from DATE, p_date_to DATE) RETURN SYS_RE
    v_cursor SYS_REFCURSOR;
    v_exists NUMBER;
BEGIN
    SELECT COUNT(*) INTO v_exists
    FROM TRIP t
    WHERE t.COUNTRY = p_country AND t.TRIP_DATE BETWEEN p_date_from AND p_date_to;

    IF v_exists = 0 THEN
        RAISE_APPLICATION_ERROR(-20001, 'There are no trips to this country');
    END IF;

    OPEN v_cursor FOR
        SELECT trip_id, country, trip_date, trip_name, max_no_places, number_left
        FROM vw_available_trip
        WHERE country = p_country
          AND trip_date BETWEEN p_date_from AND p_date_to;
    RETURN v_cursor;
END f_available_trips_to;
```

W ostatniej funkcji dodano przykładową kontrolę argumentów przekazanych do funkcji, którą uważamy za niezbędną przy używaniu w funkcji złożonych widoków. Uważamy, że nie jest to potrzebne przy prostych funkcjach.

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_iickets`
    - 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

```
create PROCEDURE p_add_reservation(
    p_trip_id INT,
    p_person_id INT,
    p_no_tickets INT) AS
    v_available_places INT;
BEGIN
    SELECT no_available_places INTO v_available_places FROM vw_trip WHERE trip_id = p_trip_id;

    IF v_available_places IS NULL THEN
        RAISE_APPLICATION_ERROR(-20001, 'Nie znaleziono wycieczki o podanym ID.');
    END IF;

    IF v_available_places < p_no_tickets 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);

    COMMIT;
END p_add_reservation;


create PROCEDURE p_modify_reservation_status(
    p_reservation_id INT,
    p_status CHAR) AS
    v_old_status CHAR;
BEGIN
    SELECT status INTO v_old_status FROM vw_reservation WHERE reservation_id = p_reservation_id;

    IF v_old_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)
    VALUES (p_reservation_id, SYSDATE, p_status);

    COMMIT;
END p_modify_reservation_status;
```

```
CREATE OR REPLACE PROCEDURE p_modify_reservation(
    reservation_id IN NUMBER,
    no_tickets IN NUMBER
) AS
    v_trip_id NUMBER;
    v_old_no_tickets NUMBER;
    v_old_status VARCHAR(1);
    v_new_status VARCHAR(1);
    v_max_places NUMBER;
    v_available_places NUMBER;
BEGIN

    IF no_tickets < 0 THEN
        RAISE_APPLICATION_ERROR(-20001, 'The no_tickets have to be greater than 0!');
    end if;

    SELECT r.TRIP_ID, r.NO_TICKETS, r.STATUS
    INTO v_trip_id, v_old_no_tickets, v_old_status
    FROM RESERVATION r
    WHERE r.reservation_id = p_modify_reservation.reservation_id;

    IF v_trip_id IS NULL THEN
        RAISE_APPLICATION_ERROR(-20002, 'Such a reservation does not exist!');
    END IF;

    SELECT NO_AVAILABLE_PLACES, MAX_NO_PLACES
    INTO v_available_places, v_max_places
    FROM VW_TRIP vwt
    WHERE vwt.TRIP_ID = v_trip_id;

    IF NO_TICKETS > v_available_places + v_old_no_tickets THEN
        RAISE_APPLICATION_ERROR(-20003, 'There are not that number of available places');
    END IF;

    IF no_tickets = 0 THEN
        v_new_status := 'C';
    ELSIF no_tickets = v_old_no_tickets THEN
        v_new_status := v_old_status;
    ELSE
        v_new_status := 'N'; -- C,N,P -> N no_tickets > 0
    END IF;

    UPDATE RESERVATION
    SET NO_TICKETS = p_modify_reservation.no_tickets,
        STATUS = v_new_status
    WHERE RESERVATION_ID = p_modify_reservation.reservation_id;

    IF V_NEW_STATUS != V_OLD_STATUS THEN
        INSERT INTO LOG(RESERVATION_ID, LOG_DATE, STATUS, NO_TICKETS)
        VALUES(p_modify_reservation.reservation_id, SYSDATE, V_NEW_STATUS, NO_TICKETS);
    END IF;
```

```
    COMMIT;
END;
```

Teoretycznie jeśli mamy status 'P' to łatwiej byłoby wycofać transakcję, inaczej trzeba by kontrolować ile ktoś musi dopłacić/dostać zwrotu, aczkolwiek jest to wykonalne i można to odnaleźć w logach, po id rezerwacji (ile biletów zostało już opłaconych itp.) .

```sql
CREATE OR REPLACE PROCEDURE p_modify_max_no_places(
    trip_id IN NUMBER,
    max_no_places IN NUMBER
) AS
    v_no_reserved_places NUMBER;
    v_trip_exists NUMBER;
BEGIN

    SELECT COUNT(*) INTO v_trip_exists
    FROM TRIP t
    WHERE t.trip_id = p_modify_max_no_places.trip_id;

    IF v_trip_exists = 0 THEN
        RAISE_APPLICATION_ERROR(-20001, 'Trip does not exist');
    END IF;

    IF p_modify_max_no_places.max_no_places IS NULL THEN
        RAISE_APPLICATION_ERROR(-20002, 'New number of places cannot be null!');
    END IF;

    IF p_modify_max_no_places.max_no_places <= 0 THEN
        RAISE_APPLICATION_ERROR(-20003, 'New number of places have to be greater than 0!');
    END IF;



    SELECT SUM(r.no_tickets)
    INTO v_no_reserved_places
    FROM reservation r
    WHERE r.trip_id = p_modify_max_no_places.trip_id AND status IN ('N', 'P');

    IF (max_no_places < v_no_reserved_places) THEN
        RAISE_APPLICATION_ERROR(-20004, 'New number of places is less than number of reserved places');
    END IF;

    UPDATE TRIP
    SET TRIP.max_no_places = p_modify_max_no_places.max_no_places
    WHERE TRIP.trip_id = p_modify_max_no_places.trip_id;

    DBMS_OUTPUT.PUT_LINE('Zaktualizowano max_no_places');
    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

Triggery

```
CREATE OR REPLACE TRIGGER trg_log_insert_reservation
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;
```

```
CREATE OR REPLACE TRIGGER trg_log_update_status
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;
```

```
CREATE OR REPLACE TRIGGER trg_update_log_no_tickets
    AFTER UPDATE OF no_tickets
    ON reservation
    FOR EACH ROW
    WHEN (NEW.no_tickets <> OLD.no_tickets)
BEGIN
    INSERT INTO log(log_id, reservation_id, log_date, status, no_tickets)
    VALUES (s_log_seq.nextval, :NEW.reservation_id, SYSDATE, :NEW.status, :NEW.no_tickets);
end;
```

```
CREATE OR REPLACE TRIGGER trg_prevent_delete_reservation
    BEFORE DELETE
    ON reservation
    FOR EACH ROW
BEGIN
    RAISE_APPLICATION_ERROR(-20001, 'You cannot delete reservation. You can only cancel it.');
end;
```

Procedury

```
CREATE OR REPLACE PROCEDURE p_add_reservation_4(
    p_trip_id INT,
    p_person_id INT,
    p_no_tickets INT) AS
    v_available_places INT;
BEGIN

    SELECT no_available_places INTO v_available_places FROM vw_trip WHERE trip_id = p_trip_id;

    IF v_available_places IS NULL THEN
        RAISE_APPLICATION_ERROR(-20001, 'Nie znaleziono wycieczki o podanym ID.');
    END IF;

    IF v_available_places < p_no_tickets 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);

    COMMIT;
END p_add_reservation_4;


CREATE OR REPLACE PROCEDURE p_modify_reservation_status_4(
    p_reservation_id INT,
    p_status CHAR) AS
    v_old_status CHAR;
BEGIN

    SELECT status INTO v_old_status FROM vw_reservation WHERE reservation_id = p_reservation_id;

    IF v_old_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;

    COMMIT;
END p_modify_reservation_status_4;
```

```
CREATE OR REPLACE PROCEDURE p_modify_reservation_4(
    reservation_id IN NUMBER,
    no_tickets IN NUMBER
) AS
    v_trip_id          NUMBER;
    v_old_no_tickets   NUMBER;
    v_old_status       VARCHAR(1);
    v_new_status       VARCHAR(1);
    v_max_places       NUMBER;
    v_available_places NUMBER;
BEGIN

    IF no_tickets < 0 THEN
        RAISE_APPLICATION_ERROR(-20001, 'The no_tickets have to be greater than 0!');
    end if;

    SELECT r.TRIP_ID, r.NO_TICKETS, r.STATUS
    INTO v_trip_id, v_old_no_tickets, v_old_status
    FROM RESERVATION r
    WHERE r.reservation_id = p_modify_reservation_4.reservation_id;

    IF v_trip_id IS NULL THEN
        RAISE_APPLICATION_ERROR(-20002, 'Such a reservation does not exist!');
    END IF;

    SELECT NO_AVAILABLE_PLACES, MAX_NO_PLACES
    INTO v_available_places, v_max_places
    FROM VW_TRIP vwt
    WHERE vwt.TRIP_ID = v_trip_id;

    IF NO_TICKETS > v_available_places + v_old_no_tickets THEN
        RAISE_APPLICATION_ERROR(-20003, 'There are not that number of available places');
    END IF;

    IF no_tickets = 0 THEN
        v_new_status := 'C';
    ELSIF no_tickets = v_old_no_tickets THEN
        v_new_status := v_old_status;
    ELSE
        v_new_status := 'N'; -- C,N,P -> N no_tickets > 0
    END IF;

    UPDATE RESERVATION
    SET NO_TICKETS = p_modify_reservation_4.no_tickets,
        STATUS     = v_new_status
    WHERE RESERVATION_ID = p_modify_reservation_4.reservation_id;

    COMMIT;
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

Triggery

```
CREATE OR REPLACE TRIGGER trg_check_availability_on_status_change
BEFORE UPDATE OF status ON reservation
FOR EACH ROW
WHEN (NEW.status = 'P' AND OLD.status != 'P')
DECLARE
    v_available_places INT;
BEGIN
    -- Pobranie liczby dostępnych miejsc z widoku vw_trip
    SELECT no_available_places INTO v_available_places
    FROM vw_trip
    WHERE trip_id = :NEW.trip_id;

    IF v_available_places < :NEW.no_tickets THEN
        RAISE_APPLICATION_ERROR(-20010, 'Nie ma wystarczającej liczby wolnych miejsc na tę wycieczkę.');
    END IF;
END;
```

```
CREATE OR REPLACE TRIGGER trg_check_availability_on_status_change
BEFORE UPDATE OF status ON reservation
FOR EACH ROW
WHEN (NEW.status = 'P' AND OLD.status != 'P')
DECLARE
    v_max_places INT;
    v_reserved_places INT;
BEGIN

    SELECT max_no_places INTO v_max_places
    FROM trip
    WHERE trip_id = :NEW.trip_id;

    SELECT COALESCE(SUM(no_tickets), 0) INTO v_reserved_places
    FROM reservation
    WHERE trip_id = :NEW.trip_id AND status = 'P' AND reservation_id != :NEW.reservation_id;

    IF (v_max_places - v_reserved_places) < :NEW.no_tickets THEN
        RAISE_APPLICATION_ERROR(-20010, 'Brak wolnych miejsc na tę wycieczkę.');
    END IF;
END;
```

```
CREATE OR REPLACE TRIGGER trg_check_reservation_availability
    BEFORE INSERT ON reservation
    FOR EACH ROW
DECLARE
    v_available_places NUMBER;
BEGIN

    SELECT NO_AVAILABLE_PLACES INTO v_available_places
    FROM VW_TRIP
    WHERE VW_TRIP.TRIP_ID = :NEW.trip_id;

    IF :NEW.no_tickets > v_available_places THEN
        RAISE_APPLICATION_ERROR(-20020, 'A lack of places! Available: ' || v_available_places);
    END IF;
END;



CREATE OR REPLACE TRIGGER trg_change_no_tickets
    BEFORE UPDATE OF no_tickets ON reservation
    FOR EACH ROW
DECLARE
    old_no_tickets NUMBER;
    new_no_tickets NUMBER;
    v_available_places NUMBER;
BEGIN
    old_no_tickets := :OLD.no_tickets;
    new_no_tickets := :NEW.no_tickets;

    SELECT NO_AVAILABLE_PLACES
    INTO v_available_places
    FROM VW_TRIP vwt
    WHERE vwt.TRIP_ID = :NEW.trip_id;

    IF new_no_tickets > v_available_places + old_no_tickets THEN
        RAISE_APPLICATION_ERROR(-20003, 'There are not that number of available places');
    END IF;
end;
```

Procedury

```
CREATE OR REPLACE PROCEDURE p_add_reservation_5(
    p_trip_id IN NUMBER,
    p_person_id IN NUMBER,
    p_no_tickets IN NUMBER
) AS
    v_person_exists NUMBER;
    v_trip_exists NUMBER;
BEGIN

    IF p_no_tickets <= 0 THEN
        RAISE_APPLICATION_ERROR(-20010, 'The number of tickets has to be greater than 0.');
    END IF;

    SELECT COUNT(*) INTO v_person_exists FROM person WHERE person_id = p_person_id;
    IF v_person_exists = 0 THEN
        RAISE_APPLICATION_ERROR(-20011, 'That person does not exist!');
    END IF;

    SELECT COUNT(*) INTO v_trip_exists FROM trip WHERE trip_id = p_trip_id;
    IF v_trip_exists = 0 THEN
        RAISE_APPLICATION_ERROR(-20012, 'The trip does not exist!');
    END IF;

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

    COMMIT;
END;


CREATE OR REPLACE PROCEDURE p_modify_reservation_status_5(
    p_reservation_id INT,
    p_status CHAR) AS
    v_old_status CHAR;
BEGIN

    SELECT status INTO v_old_status FROM reservation WHERE reservation_id = p_reservation_id;

    IF v_old_status = 'C' THEN
        RAISE_APPLICATION_ERROR(-20011, 'Nie można zmienić statusu anulowanej rezerwacji.');
    END IF;

    UPDATE reservation
    SET status = p_status
    WHERE reservation_id = p_reservation_id;

    COMMIT;
END p_modify_reservation_status_5;
```

```
CREATE OR REPLACE PROCEDURE p_modify_reservation_5(
    reservation_id IN NUMBER,
    no_tickets IN NUMBER
) AS
    v_trip_id          NUMBER;
    v_old_no_tickets   NUMBER;
    v_old_status       VARCHAR(1);
    v_new_status       VARCHAR(1);
BEGIN

    IF no_tickets < 0 THEN
        RAISE_APPLICATION_ERROR(-20001, 'The no_tickets have to be greater than 0!');
    end if;

    SELECT r.TRIP_ID, r.NO_TICKETS, r.STATUS
    INTO v_trip_id, v_old_no_tickets, v_old_status
    FROM RESERVATION r
    WHERE r.reservation_id = p_modify_reservation_5.reservation_id;

    IF v_trip_id IS NULL THEN
        RAISE_APPLICATION_ERROR(-20002, 'Such a reservation does not exist!');
    END IF;

    IF no_tickets = 0 THEN
        v_new_status := 'C';
    ELSIF no_tickets = v_old_no_tickets THEN
        v_new_status := v_old_status;
    ELSE
        v_new_status := 'N'; -- C,N,P -> N no_tickets > 0
    END IF;

    UPDATE RESERVATION
    SET NO_TICKETS = p_modify_reservation_5.no_tickets,
        STATUS     = v_new_status
    WHERE RESERVATION_ID = p_modify_reservation_5.reservation_id;

    COMMIT;
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

```
ALTER TABLE trip ADD no_available_places INT NULL;

CREATE OR REPLACE PROCEDURE p_recalculate_available_places
AS
    BEGIN
        UPDATE TRIP t
        SET no_available_places = t.max_no_places - COALESCE(
            (SELECT sum(r.no_tickets)
            FROM RESERVATION r
            WHERE r.trip_id = t.trip_id
              AND r.status IN ('N', 'P'))
            , 0);

        COMMIT;

        DBMS_OUTPUT.PUT_LINE('The available number of places have been recalculated.');
    end;


BEGIN
    P_RECALCULATE_AVAILABLE_PLACES();
end;
```

Zadanie 6a - procedury

Obsługę pola `no_available_places` należy zrealizować przy pomocy procedur

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

> UWAGA
> Należy stworzyć nowe wersje tych widoków/procedur/triggerów (np. dodając do

> nazwy dopisek 6a - od numeru zadania). Poprzednie wersje procedur należy pozostawić w celu umożliwienia weryfikacji ich poprawności.

- może być potrzebne wyłączenie 'poprzednich wersji' triggerów

Zadanie 6a - rozwiązanie

Należy wyłączyć poprzednie triggery reagujące na zmianę dostępności miejsc. W zad 6b będą nowe wersje tych triggerów.

```
CREATE OR REPLACE PROCEDURE p_add_reservation_6a(
    p_trip_id INT,
    p_person_id INT,
    p_no_tickets INT) AS
    v_available_places INT;
BEGIN

    SELECT no_available_places INTO v_available_places FROM trip WHERE trip_id = p_trip_id;

    IF v_available_places IS NULL THEN
        RAISE_APPLICATION_ERROR(-20001, 'Wycieczka nie istnieje.');
    END IF;

    IF v_available_places < p_no_tickets 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, 'P', p_no_tickets); --

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

    COMMIT;
END p_add_reservation_6a;
```

```
CREATE OR REPLACE PROCEDURE p_modify_reservation_status_6a(
    p_reservation_id INT,
    p_status CHAR) AS
    v_old_status CHAR;
    v_trip_id INT;
    v_no_tickets INT;
BEGIN
    SELECT status, trip_id, no_tickets INTO v_old_status, v_trip_id, v_no_tickets
    FROM reservation
    WHERE reservation_id = p_reservation_id;

    IF v_old_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;

    IF v_old_status != 'P' AND p_status = 'P' THEN
        UPDATE trip
        SET no_available_places = no_available_places - v_no_tickets
        WHERE trip_id = v_trip_id;
    ELSIF v_old_status = 'P' AND p_status != 'P' THEN
        UPDATE trip
        SET no_available_places = no_available_places + v_no_tickets
        WHERE trip_id = v_trip_id;
    END IF;

    COMMIT;
END p_modify_reservation_status_6a;
```

```
create or replace PROCEDURE p_modify_reservation_6a(
    reservation_id IN NUMBER,
    no_tickets IN NUMBER
) AS
    v_trip_id          NUMBER;
    v_old_no_tickets   NUMBER;
    v_old_status       VARCHAR(1);
    v_new_status       VARCHAR(1);
    v_max_places       NUMBER;
    v_available_places NUMBER;
    v_trip_date        DATE;
BEGIN

    IF no_tickets < 0 THEN
        RAISE_APPLICATION_ERROR(-20001, 'The no_tickets have to be greater than 0!');
    end if;

    SELECT r.TRIP_ID, r.NO_TICKETS, r.STATUS, t.trip_date
    INTO v_trip_id, v_old_no_tickets, v_old_status, v_trip_date
    FROM RESERVATION r
             JOIN TRIP t ON r.TRIP_ID = t.TRIP_ID
    WHERE r.reservation_id = p_modify_reservation_6a.reservation_id;

    IF v_trip_id IS NULL THEN
        RAISE_APPLICATION_ERROR(-20002, 'Such a reservation does not exist!');
    END IF;

    IF v_trip_date <= SYSDATE THEN
        RAISE_APPLICATION_ERROR(-20003, 'That trip has already taken place, cannot modify details of reservation!')
    END IF;

    SELECT NO_AVAILABLE_PLACES, MAX_NO_PLACES
    INTO v_available_places, v_max_places
    FROM VW_TRIP vwt
    WHERE vwt.TRIP_ID = v_trip_id;

    IF NO_TICKETS > v_available_places + v_old_no_tickets THEN
        RAISE_APPLICATION_ERROR(-20003, 'There are not that number of available places');
    END IF;

    IF no_tickets = 0 THEN
        v_new_status := 'C';
    ELSIF no_tickets = v_old_no_tickets THEN
        v_new_status := v_old_status;
        COMMIT;
    ELSE
        v_new_status := 'N';
    END IF;

    UPDATE RESERVATION
    SET NO_TICKETS = p_modify_reservation_6a.no_tickets,
        STATUS     = v_new_status
```

```
    WHERE RESERVATION_ID = p_modify_reservation_6a.reservation_id;

    UPDATE TRIP t
    SET no_available_places = t.max_no_places - COALESCE(
            (SELECT sum(r.no_tickets)
             FROM RESERVATION r
             WHERE r.trip_id = v_trip_id
               AND r.status IN ('N', 'P'))
        , 0)
    WHERE trip_id = v_trip_id;

    COMMIT;
END;
/
```

```
create PROCEDURE p_modify_max_no_places_6a(
    trip_id IN NUMBER,
    max_no_places IN NUMBER
) AS
    v_no_reserved_places NUMBER;
    v_trip_exists        NUMBER;
    v_trip_date DATE;
BEGIN

    SELECT COUNT(*), t.trip_date
    INTO v_trip_exists, v_trip_date
    FROM TRIP t
    WHERE t.trip_id = p_modify_max_no_places_6a.trip_id
    GROUP BY t.trip_date;
    --dla nieistniejącej wycieczki COUNT(*) powoduje że trip_date jest NULL
    -- i wyjątek nie poleci, jeśli nie użylibyśmy f. agregującem to wtedy
    -- mamy DATA_NOT_FOUND exception :)

    IF v_trip_exists = 0 THEN
        RAISE_APPLICATION_ERROR(-20001, 'Trip does not exist');
    END IF;

    IF p_modify_max_no_places_6a.max_no_places IS NULL THEN
        RAISE_APPLICATION_ERROR(-20002, 'New number of places cannot be null!');
    END IF;

    IF p_modify_max_no_places_6a.max_no_places <= 0 THEN
        RAISE_APPLICATION_ERROR(-20003, 'New number of places have to be greater than 0!');
    END IF;

    IF v_trip_date <= SYSDATE THEN
        RAISE_APPLICATION_ERROR(-20005, 'Nie można modyfikować wycieczki, która już się odbyła');
    END IF;


    SELECT SUM(r.no_tickets)
    INTO v_no_reserved_places
    FROM reservation r
    WHERE r.trip_id = p_modify_max_no_places_6a.trip_id
      AND status IN ('N', 'P');

    IF (max_no_places < v_no_reserved_places) THEN
        RAISE_APPLICATION_ERROR(-20004, 'New number of places is less than number of reserved places');
    END IF;

    UPDATE TRIP t
    SET t.max_no_places = p_modify_max_no_places_6a.max_no_places,
        t.no_available_places = t.max_no_places - COALESCE(
            (SELECT sum(r.no_tickets)
            FROM RESERVATION r
            WHERE r.trip_id = p_modify_max_no_places_6a.trip_id
              AND r.status IN ('N', 'P'))
```

```
            , 0)
    WHERE t.trip_id = p_modify_max_no_places_6a.trip_id;


    DBMS_OUTPUT.PUT_LINE('Filed max_no_places has been updated');
    COMMIT;
END;
/


--- modyfikacja widoku, zakładamy, że dzięki procedurom mamy aktualne dane w bazie:

create or replace view vw_available_trip_6a
as
select t.trip_id,
       t.trip_name,
       t.country,
       t.trip_date,
       t.max_no_places,
       t.no_available_places
from TRIP t
where t.trip_date > SYSDATE and t.no_available_places > 0;
```

Zadanie 6b - triggery

Obsługę pola `no_available_places` należy zrealizować przy pomocy triggerów

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

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

- może być potrzebne wyłączenie 'poprzednich wersji' triggerów

Zadanie 6b - rozwiązanie

```
CREATE OR REPLACE PROCEDURE p_add_reservation_6b(
    p_trip_id INT,
    p_person_id INT,
    p_no_tickets INT) AS
    v_available_places INT;
BEGIN

    SELECT no_available_places INTO v_available_places FROM trip WHERE trip_id = p_trip_id;

    IF v_available_places IS NULL THEN
        RAISE_APPLICATION_ERROR(-20001, 'Wycieczka nie istnieje.');
    END IF;

    IF v_available_places < p_no_tickets THEN
        RAISE_APPLICATION_ERROR(-20002, 'Brak wolnych miejsc.');
    END IF;

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

    COMMIT;
END p_add_reservation_6b;


REATE OR REPLACE PROCEDURE p_modify_reservation_status_6b(
    p_reservation_id INT,
    p_status CHAR) AS
    v_old_status CHAR;
BEGIN
    SELECT status INTO v_old_status FROM reservation WHERE reservation_id = p_reservation_id;

    IF v_old_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;

    COMMIT;
END p_modify_reservation_status_6b;
```

```
CREATE OR REPLACE PROCEDURE p_modify_reservation_6b(
    reservation_id IN NUMBER,
    no_tickets IN NUMBER
) AS
    v_trip_id          NUMBER;
    v_old_no_tickets   NUMBER;
    v_old_status       VARCHAR2(1);
    v_new_status       VARCHAR2(1);
    v_trip_date        DATE;
    v_available_places NUMBER;
    v_max_places       NUMBER;
BEGIN
    IF no_tickets < 0 THEN
        RAISE_APPLICATION_ERROR(-20001, 'The no_tickets have to be greater than 0!');
    END IF;


    SELECT r.TRIP_ID, r.NO_TICKETS, r.STATUS, t.trip_date
    INTO v_trip_id, v_old_no_tickets, v_old_status, v_trip_date
    FROM RESERVATION r
             JOIN TRIP t ON r.TRIP_ID = t.TRIP_ID
    WHERE r.reservation_id = p_modify_reservation_6b.reservation_id;

    IF v_trip_date <= SYSDATE THEN
        RAISE_APPLICATION_ERROR(-20003, 'That trip has already taken place, cannot modify details of reservation!')
    END IF;

    SELECT NO_AVAILABLE_PLACES, MAX_NO_PLACES
    INTO v_available_places, v_max_places
    FROM VW_TRIP vwt
    WHERE vwt.TRIP_ID = v_trip_id;

    IF NO_TICKETS > v_available_places + v_old_no_tickets THEN
        RAISE_APPLICATION_ERROR(-20003, 'There are not that number of available places');
    END IF;


    IF no_tickets = 0 THEN
        v_new_status := 'C';
    ELSIF no_tickets = v_old_no_tickets THEN
        v_new_status := v_old_status;
    ELSE
        v_new_status := 'N';
    END IF;


    UPDATE reservation
    SET no_tickets = p_modify_reservation_6b.no_tickets,
        status     = v_new_status
    WHERE reservation_id = p_modify_reservation_6b.reservation_id;

    COMMIT;
```

```
END;
/


CREATE OR REPLACE TRIGGER trg_update_no_available_on_insert_6b
AFTER INSERT ON reservation
FOR EACH ROW
WHEN (NEW.status = 'P')
BEGIN
    UPDATE trip
    SET no_available_places = no_available_places - :NEW.no_tickets
    WHERE trip_id = :NEW.trip_id;
END;


CREATE OR REPLACE TRIGGER trg_update_no_available_on_status_change_6b
AFTER UPDATE OF status ON reservation
FOR EACH ROW
BEGIN
    IF :OLD.status != 'P' AND :NEW.status = 'P' THEN

        UPDATE trip
        SET no_available_places = no_available_places - :NEW.no_tickets
        WHERE trip_id = :NEW.trip_id;
    ELSIF :OLD.status = 'P' AND :NEW.status != 'P' THEN

        UPDATE trip
        SET no_available_places = no_available_places + :NEW.no_tickets
        WHERE trip_id = :NEW.trip_id;
    END IF;
END;


CREATE OR REPLACE TRIGGER tr_reservation_manage_6b
    AFTER UPDATE ON reservation
    FOR EACH ROW
DECLARE
    v_trip_date DATE;
    v_max_places NUMBER;
BEGIN
    UPDATE TRIP t
    SET no_available_places = t.max_no_places - COALESCE(
            (SELECT sum(r.no_tickets)
             FROM RESERVATION r
             WHERE r.trip_id = :NEW.trip_id
               AND r.status IN ('N', 'P'))
        , 0)
    WHERE trip_id = :NEW.trip_id;
END;
/
```

Zadanie 7 - podsumowanie

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

Oracle PL/SQL i MS SQL Server T-SQL różnią się trochę składnią, przede wszystkim z ';' na końcu każdego polecenia oraz dostępnymi funkcjami specyficznymi dla każdego systemu. W PL/SQL bloki kodu są bardziej ustandaryzowane, podczas gdy T-SQL oferuje większą elastyczność w niektórych operacjach. Zarówno PL/SQL, jak i T-SQL wspierają kursory, procedury składowane i triggery, ale różnią się w implementacji zaawansowanych funkcji, takich jak obsługa wyjątków (TRY-CATCH vs WHEN ... THEN). W PL/SQL obowiązkowe są bloki BEGIN-END nawet dla prostych procedur, natomiast w T-SQL można po prostu użyć EXEC. PL/SQL oferuje specjalne konstrukcje jak %TYPE i %ROWTYPE w przeciwieństwie do T-SQL gdzie używa się TABLE variable. DBMS_OUTPUT wymaga specjalnej aktywacji, podczas gdy PRINT w T-SQL działa od razu. PL/SQL używa := do przypisania, T-SQL używa = lub SET z połączeniem z DECLARE.