Oracle PL/Sql

widoki, funkcje, procedury, triggery ćwiczenie

Imiona i nazwiska autorów : Paweł Gadomski, Jakub Stachecki

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);
```

```sql
-- 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`

```
-- przyklady, kod, zrzuty ekranów, komentarz ...

ALTER TABLE reservation ADD no_tickets INT DEFAULT 1 NOT NULL;

ALTER TABLE log add no_tickets INT DEFAULT 1 NOT NULL;
commit;
```

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

```
-- wyniki, kod, zrzuty ekranów, komentarz ...
-- Widok rezerwacja
create 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 trip t
         JOIN reservation r ON t.trip_id = r.trip_id
         JOIN person p ON r.person_id = p.person_id

-- Widok trip
create view VW_TRIP as
SELECT
    t.trip_id,
    t.country,
    t.trip_date,
    t.TRIP_NAME,
    t.max_no_places,
    t.max_no_places - 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 available trip
create 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_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

```
-- funkcja f_trip_patricipants
create FUNCTION f_trip_participants (p_trip_id IN NUMBER)
    RETURN sys_refcursor AS v_cursor sys_refcursor;
    BEGIN
        OPEN v_cursor FOR
            SELECT  *
            FROM vw_reservation vr
            WHERE vr.trip_id = p_trip_id AND vr.status != 'C';

    RETURN v_cursor;
    END f_trip_participants;
/

-- funkcja f_person_reservations
create FUNCTION f_person_reservations (
    p_person_id NUMBER
) RETURN SYS_REFCURSOR IS
    v_cursor SYS_REFCURSOR;
BEGIN
    OPEN v_cursor FOR
    SELECT * FROM VW_RESERVATION vr WHERE vr.PERSON_ID = p_person_id;

    return v_cursor;
END f_person_reservations;
/
```

```
--funkcja f_available_trips_to
create FUNCTION f_available_trips_to (p_country IN VARCHAR2, p_date_from IN DATE,p_date_to IN DATE )
    RETURN SYS_REFCURSOR IS v_cursor SYS_REFCURSOR;
    BEGIN
        IF p_date_from > p_date_to THEN
            RETURN NULL;
        END IF;
        OPEN v_cursor FOR
            SELECT * FROM TRIP
                WHERE TRIP.COUNTRY = p_country
                AND TRIP.TRIP_DATE BETWEEN p_date_from AND p_date_to AND TRIP.MAX_NO_PLACES > 0;
        RETURN v_cursor;

    END f_available_trips_to;
/
```

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 IN Number,
```

```
    p_person_id IN Number,
    p_no_tickets IN Number
) IS
    v_id               NUMBER;
    v_start_date       DATE;
    v_available_places NUMBER;
    v_reservation_id   NUMBER;
BEGIN
    if F_TRIP_EXISTS(p_trip_id) = 0 THEN
        RAISE_APPLICATION_ERROR(-20001, 'Taka wycieczka nie istnieje');
    end if;

    if F_PERSON_EXISTS(p_person_id) = 0 THEN
        RAISE_APPLICATION_ERROR(-20001, 'Taka osoba nie istnieje');
    end if;

    SELECT TRIP_ID, TRIP_DATE, NO_AVAILABLE_PLACES
    INTO v_id, v_start_date, v_available_places
    FROM VW_AVAILABLE_TRIP
    WHERE TRIP_ID = p_trip_id;

    IF v_start_date <= CURRENT_DATE THEN
        RAISE_APPLICATION_ERROR(-20001, 'Wycieczka już się odbyła');
    END IF;

    IF v_available_places < p_no_tickets OR p_no_tickets = 0 THEN
        RAISE_APPLICATION_ERROR(-20002, 'Brak wystarczającej ilości miejsc');
    END IF;

    INSERT INTO RESERVATION(trip_id, person_id, status, no_tickets)
    VALUES (p_trip_id, p_person_id, 'N', p_no_tickets)
    RETURNING RESERVATION_ID INTO v_reservation_id;

    INSERT INTO LOG(RESERVATION_ID, LOG_DATE, STATUS, NO_TICKETS) VALUES (v_reservation_id, CURRENT_DATE, 'N',
p_no_tickets);
end;
```

```
create PROCEDURE p_modify_reservation_status (
    p_reservation_id IN INT,
    p_status IN VARCHAR2
) IS
    v_current_status VARCHAR2(1);
BEGIN
    IF F_RESERVATION_EXISTS(p_reservation_id) = 0 THEN
        RAISE_APPLICATION_ERROR(-20003, 'Rezerwacja o podanym ID 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(-20004, '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
            );
end;

create PROCEDURE p_modify_reservation(
        p_reservation_id IN Number,
        p_no_tickets IN Number
    ) IS
        v_available_places NUMBER;
    BEGIN
        IF F_RESERVATION_EXISTS(p_reservation_id) = 0 THEN
            RAISE_APPLICATION_ERROR(-20003, 'Rezerwacja o podanym ID nie istnieje.');
```

```
        end if;

        SELECT NO_AVAILABLE_PLACES
        INTO v_available_places
        FROM VW_TRIP tr
        JOIN VW_RESERVATION re ON re.TRIP_ID = tr.TRIP_ID
        WHERE RESERVATION_ID = p_reservation_id;

        IF v_available_places < p_no_tickets THEN
            RAISE_APPLICATION_ERROR(-20003, 'Brak wystarczającej ilości wolnych miejsc');
        end if;

        UPDATE RESERVATION
        SET NO_TICKETS = p_no_tickets
        WHERE RESERVATION_ID = p_reservation_id;

        UPDATE LOG
        SET NO_TICKETS = p_no_tickets
        WHERE RESERVATION_ID = p_reservation_id;
    end;

  create PROCEDURE p_modify_max_number_of_places (
    p_trip_id IN INT,
    p_max_no_of_places IN INT
) IS
    v_already_reserved_count INT;
BEGIN
    IF F_TRIP_EXISTS(p_trip_id) = 0 THEN
        RAISE_APPLICATION_ERROR(-20003, 'Rezerwacja o podanym ID nie istnieje.');
    end if;
    SELECT NVL(MAX_NO_PLACES-NO_AVAILABLE_PLACES,0) INTO v_already_reserved_count
    FROM VW_AVAILABLE_TRIP
    WHERE TRIP_ID=p_trip_id;

    IF p_max_no_of_places < v_already_reserved_count THEN
        RAISE_APPLICATION_ERROR(-20004, 'Podana maksymalna liczba miejsc jest mniejsza niż liczba już zarezerwowanych
```

```
miejsc');
    end if;

    UPDATE TRIP
    SET MAX_NO_PLACES = p_max_no_of_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

```
--trigger zabraniający usunięcia rezerwacji

create trigger TR_PREVENT_RESERVATION_DELETION
    before delete
    on RESERVATION
    for each row
BEGIN
        RAISE_APPLICATION_ERROR(-20005, 'Nie można usunąć rezerwacji');
    end;
/

--trigger obsługujący dodanie rezerwacji

create trigger TR_RESERVATION_INSERT
    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 obługujący zmianę statusu rezerwacji oraz liczby zarezerwowanych/kupionych biletów

create trigger TR_RESERVATION_STATUS_CHANGE
    after update
    on RESERVATION
    for each row
    when (OLD.STATUS <> NEW.STATUS OR 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;
/

--zmodyfikowane procedury

create PROCEDURE p_add_reservation_4(
    p_trip_id IN Number,
    p_person_id IN Number,
    p_no_tickets IN Number
) IS
    v_id               NUMBER;
    v_start_date       DATE;
    v_available_places NUMBER;
    v_reservation_id   NUMBER;
BEGIN
    if F_TRIP_EXISTS(p_trip_id) = 0 THEN
        RAISE_APPLICATION_ERROR(-20001, 'Taka wycieczka nie istnieje');
    end if;

    if F_PERSON_EXISTS(p_person_id) = 0 THEN
        RAISE_APPLICATION_ERROR(-20001, 'Taka osoba nie istnieje');
    end if;

    SELECT TRIP_ID, TRIP_DATE, NO_AVAILABLE_PLACES
    INTO v_id, v_start_date, v_available_places
    FROM VW_AVAILABLE_TRIP
    WHERE TRIP_ID = p_trip_id;

    IF v_start_date <= CURRENT_DATE THEN
        RAISE_APPLICATION_ERROR(-20001, 'Wycieczka już się odbyła');
    END IF;

    IF v_available_places < p_no_tickets OR p_no_tickets = 0 THEN
        RAISE_APPLICATION_ERROR(-20002, 'Brak wystarczającej ilości miejsc');
```

```
    END IF;

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

end;
/

create PROCEDURE p_modify_reservation_status_4 (
    p_reservation_id IN INT,
    p_status IN VARCHAR2
) IS
    v_current_status VARCHAR2(1);
BEGIN
    IF F_RESERVATION_EXISTS(p_reservation_id) = 0 THEN
        RAISE_APPLICATION_ERROR(-20003, 'Rezerwacja o podanym ID 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(-20004, 'Nie można zmienić statusu anulowanej rezerwacji.');
    end if;

    UPDATE RESERVATION
    SET STATUS = p_status
    WHERE RESERVATION_ID = p_reservation_id;

end;
/

create PROCEDURE p_modify_reservation_4(
    p_reservation_id IN Number,
    p_no_tickets IN Number
```

```
) IS
    v_available_places NUMBER;
BEGIN
    IF F_RESERVATION_EXISTS(p_reservation_id) = 0 THEN
        RAISE_APPLICATION_ERROR(-20003, 'Rezerwacja o podanym ID nie istnieje.');
    end if;

    SELECT NO_AVAILABLE_PLACES
    INTO v_available_places
    FROM VW_TRIP tr
             JOIN VW_RESERVATION re ON re.TRIP_ID = tr.TRIP_ID
    WHERE RESERVATION_ID = p_reservation_id;

    IF v_available_places < p_no_tickets THEN
        RAISE_APPLICATION_ERROR(-20003, 'Brak wystarczającej ilości wolnych miejsc');
    end if;

    UPDATE RESERVATION
    SET NO_TICKETS = p_no_tickets
    WHERE RESERVATION_ID = p_reservation_id;

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_status_5`

Zadanie 5 - rozwiązanie

```
-- trigger obsługujący dodanie rezerwacji

create trigger TR_RESERVATION_INSERT_5
    before insert
    on RESERVATION
    for each row
DECLARE

    v_available_places NUMBER;
    v_trip_date        DATE;
    BEGIN
        IF F_TRIP_EXISTS(:NEW.TRIP_ID) = 0 THEN
            RAISE_APPLICATION_ERROR(-20005, 'Nie znaleziono wycieczki o podanym ID');
        end if;
```

```
        IF F_PERSON_EXISTS(:NEW.TRIP_ID) = 0 THEN
            RAISE_APPLICATION_ERROR(-20006, 'Nie znaleziono osoby o podanym ID');
        end if;


        SELECT NO_AVAILABLE_PLACES, TRIP_DATE INTO v_available_places, v_trip_date
        FROM VW_TRIP
        WHERE VW_TRIP.TRIP_ID = :NEW.TRIP_ID;

        IF v_available_places < :NEW.NO_TICKETS THEN
            RAISE_APPLICATION_ERROR(-20007, 'Brak wystarczającej liczby wolnych miejsc.');
        end if;

        IF v_trip_date < SYSDATE THEN
            RAISE_APPLICATION_ERROR(-20008, 'Wycieczka o podanym ID już się odbyła');
        end if;
end;

create trigger TR_RESERVATION_STATUS_CHANGE_5
    before update
    on RESERVATION
    for each row
    when (OLD.STATUS <> NEW.STATUS)
BEGIN
    IF :OLD.STATUS = 'C' THEN
        RAISE_APPLICATION_ERROR(-20004, 'Nie można zmienić statusu anulowanej rezerwacji.');
    end if;
END;

create trigger TR_RESERVATION_CHANGE_NO_TICKETS_5
    before update of NO_TICKETS
    on RESERVATION
    for each row
    when (OLD.NO_TICKETS <> NEW.NO_TICKETS)
DECLARE
```

```
    v_available_places INT;
    v_trip_date DATE;
BEGIN
    SELECT GET_AVAILABLE_PLACES(:NEW.TRIP_ID)
    INTO v_available_places
    FROM dual;

    IF :NEW.NO_TICKETS > v_available_places THEN
        RAISE_APPLICATION_ERROR(-20007, 'Niewystarczająca ilość wolnych miejsc');
    END IF;

    SELECT TRIP_DATE
    INTO v_trip_date
    FROM TRIP
    WHERE TRIP.TRIP_ID = :OLD.TRIP_ID;

    IF v_trip_date < SYSDATE THEN
        RAISE_APPLICATION_ERROR(-20007, 'Ta wycieczka już się odbyła');
    END IF;
END;

create PROCEDURE p_modify_reservation_5(
    p_reservation_id IN Number,
    p_no_tickets IN Number
) IS
BEGIN
    UPDATE RESERVATION
    SET NO_TICKETS = p_no_tickets
    WHERE RESERVATION_ID = p_reservation_id;
end;

create PROCEDURE p_modify_reservation_status_5 (
    p_reservation_id IN INT,
    p_status IN VARCHAR2
) IS
BEGIN
```

```
    UPDATE RESERVATION
    SET STATUS = p_status
    WHERE RESERVATION_ID = p_reservation_id;

EXCEPTION
    WHEN OTHERS THEN
        ROLLBACK;
        RAISE;
end;

create PROCEDURE p_add_reservation_5(
    p_trip_id IN Number,
    p_person_id IN Number,
    p_no_tickets IN Number
) AS
BEGIN

    INSERT INTO RESERVATION(trip_id, person_id, status, no_tickets)
    VALUES (p_trip_id, p_person_id, 'N', 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 update'ująca no_available_places

create PROCEDURE update_no_available_places AS
BEGIN
    UPDATE trip t
    SET no_available_places = NVL((
        SELECT t.max_no_places - COUNT(r.trip_id)
        FROM reservation r
        WHERE r.trip_id = t.trip_id
          AND r.status <> 'C'
        GROUP BY t.max_no_places
    ), 0);
END;
```

Zadanie 6a - procedury

Obsługę pola `no_available_places` należy zrealizować przy pomocy procedur

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

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

- może być potrzebne wyłączenie 'poprzednich wersji' triggerów

Zadanie 6a - rozwiązanie

```
-- procedura dodająca rezerwację

create PROCEDURE p_add_reservation_6a(
    v_trip_id IN NUMBER,
    v_person_id IN NUMBER,
    v_no_tickets IN NUMBER
) IS
BEGIN
    INSERT INTO RESERVATION(TRIP_ID, PERSON_ID, STATUS, NO_TICKETS)
    VALUES (v_trip_id, v_person_id, 'N', v_no_tickets);

    UPDATE TRIP
    SET NO_AVAILABLE_PLACES = NO_AVAILABLE_PLACES - v_no_tickets
```

```
    WHERE TRIP_ID = v_trip_id;
end;
-- procedura odpowiedzialna za zmianę statusu rezerwacji
create PROCEDURE p_modify_reservation_max_no_places_6a (
    p_trip_id IN PLS_INTEGER,
    p_max_no_places IN PLS_INTEGER
) AS
    v_trip_id PLS_INTEGER;
    v_old_max_no_places PLS_INTEGER;
    v_available_places PLS_INTEGER;
BEGIN
    IF F_TRIP_EXISTS(p_trip_id) = 0 THEN
        RAISE_APPLICATION_ERROR(-20001, 'Taka wycieczka nie istnieje');
    end if;

    SELECT TRIP_ID, MAX_NO_PLACES, no_available_places
    INTO v_trip_id, v_old_max_no_places, v_available_places
    FROM TRIP
    WHERE TRIP_ID = p_trip_id;

    IF p_max_no_places < v_old_max_no_places - v_available_places THEN
        RAISE_APPLICATION_ERROR(-20001, 'Nowa liczba miejsc nie może być niższa od obecnie dostępnych');
    end if;

    UPDATE TRIP
    SET MAX_NO_PLACES = p_max_no_places,
        TRIP.no_available_places = v_available_places + (p_max_no_places - v_old_max_no_places)
    WHERE TRIP.TRIP_ID = p_trip_id;
end;

--procedura odpowiedzialna za zmianę maksymalnej liczby miejsc na wycieczki
create PROCEDURE p_modify_reservation_max_no_places_6a (
    p_trip_id IN PLS_INTEGER,
    p_max_no_places IN PLS_INTEGER
) AS
    v_trip_id PLS_INTEGER;
```

```
    v_old_max_no_places PLS_INTEGER;
    v_available_places PLS_INTEGER;
BEGIN
    IF F_TRIP_EXISTS(p_trip_id) = 0 THEN
        RAISE_APPLICATION_ERROR(-20001, 'Taka wycieczka nie istnieje');
    end if;

    SELECT TRIP_ID, MAX_NO_PLACES, no_available_places
    INTO v_trip_id, v_old_max_no_places, v_available_places
    FROM TRIP
    WHERE TRIP_ID = p_trip_id;

    IF p_max_no_places < v_old_max_no_places - v_available_places THEN
        RAISE_APPLICATION_ERROR(-20001, 'Nowa liczba miejsc nie może być niższa od obecnie dostępnych');
    end if;

    UPDATE TRIP
    SET MAX_NO_PLACES = p_max_no_places,
        TRIP.no_available_places = v_available_places + (p_max_no_places - v_old_max_no_places)
    WHERE TRIP.TRIP_ID = p_trip_id;
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

```
create trigger TR_UPDATE_NO_AVAILABLE_PLACES_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;

CREATE TRIGGER TR_UPDATE_STATUS_6B
    AFTER UPDATE ON RESERVATION
    FOR EACH ROW
BEGIN
    IF :NEW.STATUS = 'C' THEN
        UPDATE TRIP SET NO_AVAILABLE_PLACES = NO_AVAILABLE_PLACES + :NEW.NO_TICKETS WHERE TRIP_ID = :NEW.TRIP_ID;
    end if;
end;
```

Zadanie 7 - podsumowanie

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

```
-- W PL/SQL nie mamy auto-commitów znanych z MS SQLserver
-- wszystkie transakcje musimy zatwierdzać jawnie commit/rollback
-- Dodatkowo z naszego doświadczenia wynika, że operacje takie jak
-- Włączanie/Wyłączanie triggerów oraz zmiana procedur/triggerów
-- działają bardzo ociężale w PL/SQL, możliwe, że jest to coś
-- co łatwo naprawić, ale na pewno w MS Sqlserverze nie mieliśmy
-- takich problemów
-- Ciekawym zabiegiem jest również PRAGMA AUTONOMOUS_TRANSACTION
-- czyli funkcja która jest wywoływana w osobnej transakcji
```