Oracle PL/Sql

widoki, funkcje, procedury, triggery
ćwiczenie

Imiona i nazwiska autorów : Mateusz Pawliczek. Filip Malejki

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`

Zadanie 0 - Realizacja

Zaczynamy od utworzenia kolumn no_tickets w tabelach RESERVATION i LOGS

```
alter table RESERVATION
    add NO_TICKETS NUMBER

alter table LOG
    add NO_TICKETS NUMBER
```

Do tabeli RESERVATION wstawiamy dane które dodatkowo zawierają informację o ilości zakupionych biletów NO_TICKETS

```
insert  into reservation(trip_id, person_id, status, NO_TICKETS)
values (1, 1, 'P', 2);

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

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

insert into reservation(trip_id, person_id, status, NO_TICKETS)
values (2, 4, 'C', 1);

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

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

Realizacją zadanie jest stworzenie kilku widoków, które ułatwią dostęp do danych. Każdy z widoków pełni określoną funkcję.

1. Widok `vw_reservation`

Widok `vw_reservation` łączy dane z tabel `trip`, `person` i `reservation`, aby uzyskać szczegółowe informacje o rezerwacjach, uczestnikach oraz szczegółach wycieczek.

```
CREATE VIEW vw_reservation AS
SELECT
    RESERVATION_ID, COUNTRY, TRIP_DATE, TRIP_NAME,
    FIRSTNAME, LASTNAME, STATUS, TRIP.TRIP_ID, PERSON.PERSON_ID, NO_TICKETS
FROM
    RESERVATION
    INNER JOIN PERSON ON RESERVATION.PERSON_ID = PERSON.PERSON_ID
    INNER JOIN TRIP ON RESERVATION.TRIP_ID = TRIP.TRIP_ID;
```

Opis:

- Widok łączy dane z trzech tabel (`RESERVATION`, `PERSON`, `TRIP`), zwracając szczegóły dotyczące rezerwacji, wycieczki oraz osoby, która dokonała rezerwacji.

2. Widok `vw_trip`

Widok `vw_trip` prezentuje informacje o wszystkich dostępnych wycieczkach wraz z liczbą dostępnych miejsc. Oblicza liczbę wolnych miejsc na podstawie zarezerwowanych biletów.

```sql
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 - COALESCE(SUM(R.NO_TICKETS), 0)) AS NO_AVAILABLE_PLACES
FROM
    TRIP T
    LEFT JOIN RESERVATION R ON T.TRIP_ID = R.TRIP_ID
GROUP BY
    T.TRIP_ID, T.COUNTRY, T.TRIP_DATE, T.TRIP_NAME, T.MAX_NO_PLACES;
```

Opis:

- Widok oblicza liczbę dostępnych miejsc na każdej wycieczce, uwzględniając liczbę zarezerwowanych biletów. Zwracane dane obejmują identyfikator wycieczki, nazwę, kraj, datę oraz liczbę dostępnych miejsc.

3. Widok `vw_available_trip`

Widok `vw_available_trip` wyświetla tylko te wycieczki, które mają dostępne miejsca i które jeszcze się nie odbyły.

```sql
CREATE VIEW vw_available_trip AS
SELECT
    T.TRIP_ID, T.COUNTRY, T.TRIP_DATE, T.TRIP_NAME, T.MAX_NO_PLACES,
    (T.MAX_NO_PLACES - COALESCE(SUM(R.NO_TICKETS), 0)) AS NO_AVAILABLE_PLACES
FROM
    TRIP T
    LEFT JOIN RESERVATION R ON T.TRIP_ID = R.TRIP_ID
GROUP BY
    T.TRIP_ID, T.COUNTRY, T.TRIP_DATE, T.TRIP_NAME, T.MAX_NO_PLACES
HAVING
    (T.MAX_NO_PLACES - COALESCE(SUM(R.NO_TICKETS), 0)) > 0
    AND SYSDATE < T.TRIP_DATE;
```

Opis:

- Widok filtruje wycieczki, pokazując tylko te, które są dostępne (mają wolne miejsca) oraz te, które nie odbyły się jeszcze w przeszłości. Zwracane dane obejmują te same informacje co w widoku `vw_trip` , z dodatkowym warunkiem daty.

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

Realizacja tego zadania obejmuje utworzenie funkcji korzystając z następnujących elementów:

1. Typy Obiektowe

Typ Obiektowy `trip_participant_obj`

Typ obiektowy `trip_participant_obj` reprezentuje pojedynczego uczestnika rezerwacji na wycieczce. Zawiera informacje o rezerwacji, uczestniku oraz szczegółach wycieczki.

```
CREATE OR REPLACE TYPE trip_participant_obj AS OBJECT (
    RESERVATION_ID NUMBER,
    COUNTRY VARCHAR2(100),
    TRIP_DATE DATE,
    TRIP_NAME VARCHAR2(100),
    FIRSTNAME VARCHAR2(100),
    LASTNAME VARCHAR2(100),
    STATUS VARCHAR2(1),
    TRIP_ID NUMBER,
    PERSON_ID NUMBER,
    NO_TICKETS NUMBER
);
```

Typ Tablicowy `trip_participant_table`

Typ tablicowy `trip_participant_table` to kolekcja obiektów `trip_participant_obj`, przechowująca listę uczestników rezerwacji.

```
CREATE OR REPLACE TYPE trip_participant_table AS TABLE OF trip_participant_obj;
```

Typ Obiektowy `available_trip_obj`

Typ obiektowy `available_trip_obj` reprezentuje szczegóły wycieczki oraz liczbę dostępnych miejsc na danej wycieczce.

```sql
CREATE OR REPLACE TYPE available_trip_obj AS OBJECT (
    TRIP_ID NUMBER,
    COUNTRY VARCHAR2(100),
    TRIP_DATE DATE,
    TRIP_NAME VARCHAR2(100),
    AVAILABLE_PLACES NUMBER
);
```

Typ Tablicowy `available_trip_table`

Typ tablicowy `available_trip_table` to kolekcja obiektów `available_trip_obj`, przechowująca listę dostępnych wycieczek.

```sql
CREATE OR REPLACE TYPE available_trip_table AS TABLE OF available_trip_obj;
```

2. Funkcje

Funkcja `f_trip_participants`

Funkcja `f_trip_participants` zwraca listę uczestników dla wskazanej wycieczki (`p_trip_id`). Zwracane dane obejmują szczegóły rezerwacji, osoby oraz informacje o wycieczce.

```sql
CREATE OR REPLACE FUNCTION f_trip_participants (p_trip_id NUMBER)
RETURN trip_participant_table AS
    result trip_participant_table := trip_participant_table();
BEGIN
    SELECT
        trip_participant_obj(
            RESERVATION_ID, COUNTRY, TRIP_DATE, TRIP_NAME,
            FIRSTNAME, LASTNAME, STATUS, TRIP.TRIP_ID, PERSON.PERSON_ID, NO_TICKETS
        )
    BULK COLLECT INTO result
    FROM
        RESERVATION
        INNER JOIN PERSON ON RESERVATION.PERSON_ID = PERSON.PERSON_ID
        INNER JOIN TRIP ON RESERVATION.TRIP_ID = TRIP.TRIP_ID
    WHERE
        TRIP.TRIP_ID = p_trip_id;

    RETURN result;
END;
```

Funkcja `f_person_reservations`

Funkcja `f_person_reservations` zwraca listę rezerwacji dla danej osoby (`p_person_id`). Zawiera szczegóły rezerwacji, wycieczki oraz dane osoby.

```
CREATE OR REPLACE FUNCTION f_person_reservations (p_person_id NUMBER)
RETURN trip_participant_table AS
    result trip_participant_table := trip_participant_table();
BEGIN
    SELECT
        trip_participant_obj(
            RESERVATION_ID, COUNTRY, TRIP_DATE, TRIP_NAME,
            FIRSTNAME, LASTNAME, STATUS, TRIP.TRIP_ID, PERSON.PERSON_ID, NO_TICKETS
        )
    BULK COLLECT INTO result
    FROM
        RESERVATION
        INNER JOIN PERSON ON RESERVATION.PERSON_ID = PERSON.PERSON_ID
        INNER JOIN TRIP ON RESERVATION.TRIP_ID = TRIP.TRIP_ID
    WHERE
        PERSON.PERSON_ID = p_person_id;

    RETURN result;
END;
```

Funkcja `f_available_trips_to`

Funkcja `f_available_trips_to` zwraca listę dostępnych wycieczek do wskazanego kraju (`p_country`) w zadanym okresie czasu (`p_date_from` , `p_date_to`). Oblicza liczbę dostępnych miejsc na każdej wycieczce.

```
CREATE OR REPLACE FUNCTION f_available_trips_to (
    p_country VARCHAR2,
    p_date_from DATE,
    p_date_to DATE
) RETURN available_trip_table AS
    result available_trip_table := available_trip_table();
BEGIN
    SELECT
        available_trip_obj(
            T.TRIP_ID, T.COUNTRY, T.TRIP_DATE, T.TRIP_NAME,
            (T.MAX_NO_PLACES - COALESCE(SUM(R.NO_TICKETS), 0))
        )
    BULK COLLECT INTO result
    FROM
        TRIP T
        LEFT JOIN RESERVATION R ON T.TRIP_ID = R.TRIP_ID
    WHERE
        T.COUNTRY = p_country
        AND T.TRIP_DATE BETWEEN p_date_from AND p_date_to
    GROUP BY
        T.TRIP_ID, T.COUNTRY, T.TRIP_DATE, T.TRIP_NAME, T.MAX_NO_PLACES
    HAVING
        (T.MAX_NO_PLACES - COALESCE(SUM(R.NO_TICKETS), 0)) > 0
        AND SYSDATE < T.TRIP_DATE;

    RETURN result;
END;
```

Zadanie 3 - procedury

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

Procedury

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

W ramach realizacji zadania zostały stworzone cztery procedury, które pozwalają na dodawanie, modyfikowanie statusu, zmienianie liczby miejsc rezerwacji oraz zmienianie maksymalnej liczby dostępnych miejsc na wycieczce.

1. Procedura `p_add_reservation`

Procedura `p_add_reservation` umożliwia dodanie nowej rezerwacji na wycieczkę. Zanim rezerwacja zostanie dodana, sprawdzana jest dostępność miejsc na wycieczce i czy wycieczka nie jest już zakończona.

```
CREATE OR REPLACE PROCEDURE p_add_reservation (
    p_trip_id NUMBER,
    p_person_id NUMBER,
    p_no_tickets NUMBER
) AS
    trip_date DATE;
    occupied NUMBER;
    free_spots NUMBER;
BEGIN
    BEGIN
        SELECT TRIP_DATE, MAX_NO_PLACES INTO trip_date, free_spots
        FROM TRIP WHERE TRIP.TRIP_ID = p_trip_id;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            RAISE_APPLICATION_ERROR(-20000, 'This trip does not exist...');
    END;

    IF trip_date < SYSDATE THEN
        RAISE_APPLICATION_ERROR(-20001, 'Picked an old trip...');
    END IF;

    SELECT COALESCE(SUM(RESERVATION.NO_TICKETS), 0) INTO occupied
    FROM RESERVATION
    WHERE RESERVATION.TRIP_ID = p_trip_id
    AND RESERVATION.STATUS IN ('N', 'P');

    IF occupied + p_no_tickets > free_spots THEN
        RAISE_APPLICATION_ERROR(-20002, 'Trip is full...');
    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;
```

Opis:

- Procedura pobiera ID wycieczki, ID osoby oraz liczbę biletów.
- Sprawdza, czy wycieczka istnieje oraz czy data wycieczki nie jest przeszła.
- Sprawdza dostępność miejsc na wycieczce.

- Dodaje nową rezerwację, jeżeli warunki są spełnione.

2. Procedura `p_modify_reservation_status`

Procedura `p_modify_reservation_status` umożliwia zmianę statusu rezerwacji, z dodatkowymi kontrolami dotyczącymi dostępności miejsc.

```
CREATE OR REPLACE PROCEDURE p_modify_reservation_status (
    p_reservation_id NUMBER,
    p_status VARCHAR2
) AS
    current_status VARCHAR2(1);
    trip_id NUMBER;
    trip_date DATE;
    free_spots NUMBER;
    occupied NUMBER;
BEGIN
    BEGIN
        SELECT STATUS, TRIP_ID INTO current_status, trip_id
        FROM RESERVATION
        WHERE RESERVATION_ID = p_reservation_id;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            RAISE_APPLICATION_ERROR(-20000, 'Reservation not found...');
    END;

    IF current_status = 'C' AND p_status IN ('N', 'P') THEN
        BEGIN
            SELECT TRIP_DATE, MAX_NO_PLACES INTO trip_date, free_spots
            FROM TRIP WHERE TRIP_ID = trip_id;
        EXCEPTION
            WHEN NO_DATA_FOUND THEN
                RAISE_APPLICATION_ERROR(-20001, 'Trip not found...');
        END;

        SELECT COALESCE(SUM(NO_TICKETS), 0) INTO occupied
        FROM RESERVATION
        WHERE TRIP_ID = trip_id
        AND STATUS IN ('N', 'P');

        IF occupied >= free_spots THEN
            RAISE_APPLICATION_ERROR(-20002, 'No available spots on the trip...');
        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)
    SELECT RESERVATION_ID, SYSDATE, p_status, NO_TICKETS
    FROM RESERVATION
    WHERE RESERVATION_ID = p_reservation_id;

    COMMIT;
END;
```

Opis:

- Procedura zmienia status rezerwacji (np. z "N" na "P" lub "C").
- Przed zmianą statusu sprawdza, czy wycieczka ma dostępne miejsca.
- Loguje zmianę statusu w tabeli logów.

3. Procedura `p_modify_reservation`

Procedura `p_modify_reservation` pozwala na zmianę liczby biletów w już istniejącej rezerwacji, biorąc pod uwagę dostępność miejsc na wycieczce.

```
CREATE OR REPLACE PROCEDURE p_modify_reservation (
    p_reservation_id NUMBER,
    p_no_tickets NUMBER
) AS
    p_current_no_tickets NUMBER;
    p_trip_id NUMBER;
    p_free_spots NUMBER;
    p_occupied NUMBER;
BEGIN
    BEGIN
        SELECT NO_TICKETS, TRIP_ID INTO p_current_no_tickets, p_trip_id
        FROM RESERVATION
        WHERE RESERVATION_ID = p_reservation_id;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            RAISE_APPLICATION_ERROR(-20000, 'Reservation not found...');
    END;

    SELECT MAX_NO_PLACES INTO p_free_spots
    FROM TRIP
    WHERE TRIP.TRIP_ID = p_trip_id
    AND ROWNUM = 1;

    SELECT COALESCE(SUM(NO_TICKETS), 0) INTO p_occupied
    FROM RESERVATION
    WHERE TRIP_ID = p_trip_id
    AND STATUS IN ('N', 'P');

    IF (p_occupied - p_current_no_tickets + p_no_tickets) > p_free_spots THEN
        RAISE_APPLICATION_ERROR(-20001, 'Not enough available spots on the trip...');
    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)
    SELECT RESERVATION_ID, SYSDATE, STATUS, p_no_tickets
    FROM RESERVATION
    WHERE RESERVATION_ID = p_reservation_id;

    COMMIT;
END;
/
```

Opis:

- Procedura umożliwia zmianę liczby biletów w rezerwacji.

- Przed zmianą sprawdzana jest dostępność miejsc.
- Zmiana liczby biletów jest logowana w tabeli logów.

4. Procedura `p_modify_max_no_places`

Procedura `p_modify_max_no_places` umożliwia zmianę maksymalnej liczby miejsc na wycieczce, jeśli liczba zarezerwowanych biletów nie przekracza nowej wartości.

```
CREATE OR REPLACE PROCEDURE p_modify_max_no_places (
    p_trip_id NUMBER,
    p_max_no_places NUMBER
) AS
    reserved NUMBER;
BEGIN
    SELECT COALESCE(SUM(NO_TICKETS), 0) INTO reserved
    FROM RESERVATION
    WHERE TRIP_ID = p_trip_id
    AND STATUS IN ('N', 'P');

    IF p_max_no_places < reserved THEN
        RAISE_APPLICATION_ERROR(-20000, 'Cannot reduce the number of places below the number of reserved tickets...
    END IF;

    UPDATE TRIP
    SET MAX_NO_PLACES = p_max_no_places
    WHERE TRIP_ID = p_trip_id;

    COMMIT;
END;
```

Opis:

- Procedura umożliwia zmianę maksymalnej liczby miejsc na wycieczce, ale tylko jeśli liczba zarezerwowanych biletów nie przekracza nowej liczby dostępnych miejsc.
- W przypadku próby zmniejszenia liczby miejsc poniżej liczby zarezerwowanych biletów, procedura zwróci błąd.

5. Przykładowe wywołania

- Dodanie nowej rezerwacji:

```
BEGIN
    p_add_reservation(1, 3, 2);
END;
```

- Modyfikacja statusu rezerwacji:

```
BEGIN
    p_modify_reservation_status(27, 'P');
END;
```

- Modyfikacja ilości zamówionych biletów

```
BEGIN
    p_modify_reservation(41, 5);
END;
```

- Modyfikacja maksymalnej ilości miejsc na wyjeździe

```
BEGIN
    p_modify_max_no_places(4, 50);
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 ramach zadania został stworzony wyzwalacz, które odpowiada za aktualizowanie danych w tabeli LOG. Przy każdym dodanym lub modyfikowanym rekordzie informacja o rekordzie zostaje dodane również do tabeli LOG.

Modyfikacja / Stworzenie rezerwacji

```
CREATE OR REPLACE TRIGGER T_RESERVATION_LOG
AFTER INSERT OR UPDATE 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);
    COMMIT;
END;
```

W procedurach zostały usunięte fragmenty odpowiadające za dodawanie logów. Teraz to zadanie pełnią stworzone triggery.

p_add_reservation_4

```
CREATE OR REPLACE PROCEDURE p_add_reservation_4 (
    p_trip_id NUMBER,
    p_person_id NUMBER,
    p_no_tickets NUMBER
) AS
    trip_date DATE;
    occupied NUMBER;
    free_spots NUMBER;
BEGIN
    BEGIN
        SELECT TRIP_DATE, MAX_NO_PLACES INTO trip_date, free_spots
        FROM TRIP WHERE TRIP.TRIP_ID = p_trip_id;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            RAISE_APPLICATION_ERROR(-20000, 'This trip does not exist...');
    END;

    IF trip_date < SYSDATE THEN
        RAISE_APPLICATION_ERROR(-20001, 'Picked an old trip...');
    END IF;

    SELECT COALESCE(SUM(RESERVATION.NO_TICKETS), 0) INTO occupied
    FROM RESERVATION
    WHERE RESERVATION.TRIP_ID = p_trip_id
    AND RESERVATION.STATUS IN ('N', 'P');

    IF occupied + p_no_tickets > free_spots THEN
        RAISE_APPLICATION_ERROR(-20002, 'Trip is full...');
    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_modify_reservation_status_4

```
CREATE OR REPLACE PROCEDURE p_modify_reservation_status_4 (
    p_reservation_id NUMBER,
    p_status VARCHAR2
) AS
    current_status VARCHAR2(1);
    trip_id NUMBER;
    trip_date DATE;
    free_spots NUMBER;
    occupied NUMBER;
BEGIN
    BEGIN
        SELECT STATUS, TRIP_ID INTO current_status, trip_id
        FROM RESERVATION
        WHERE RESERVATION_ID = p_reservation_id;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            RAISE_APPLICATION_ERROR(-20000, 'Reservation not found...');
    END;

    IF current_status = 'C' AND p_status IN ('N', 'P') THEN
        BEGIN
            SELECT TRIP_DATE, MAX_NO_PLACES INTO trip_date, free_spots
            FROM TRIP WHERE TRIP_ID = trip_id;
        EXCEPTION
            WHEN NO_DATA_FOUND THEN
                RAISE_APPLICATION_ERROR(-20001, 'Trip not found...');
        END;

        SELECT COALESCE(SUM(NO_TICKETS), 0) INTO occupied
        FROM RESERVATION
        WHERE TRIP_ID = trip_id
        AND STATUS IN ('N', 'P');

        IF occupied >= free_spots THEN
            RAISE_APPLICATION_ERROR(-20002, 'No available spots on the trip...');
        END IF;
    END IF;

    UPDATE RESERVATION
    SET STATUS = p_status
    WHERE RESERVATION_ID = p_reservation_id;

    COMMIT;
END;
```

p_modify_reservation_4

```
CREATE OR REPLACE PROCEDURE p_modify_reservation_4 (
    p_reservation_id NUMBER,
    p_no_tickets NUMBER
) AS
    p_current_no_tickets NUMBER;
    p_trip_id NUMBER;
    p_free_spots NUMBER;
    p_occupied NUMBER;
BEGIN
    BEGIN
        SELECT NO_TICKETS, TRIP_ID INTO p_current_no_tickets, p_trip_id
        FROM RESERVATION
        WHERE RESERVATION_ID = p_reservation_id;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            RAISE_APPLICATION_ERROR(-20000, 'Reservation not found...');
    END;

    SELECT MAX_NO_PLACES INTO p_free_spots
    FROM TRIP
    WHERE TRIP.TRIP_ID = p_trip_id
    AND ROWNUM = 1;

    SELECT COALESCE(SUM(NO_TICKETS), 0) INTO p_occupied
    FROM RESERVATION
    WHERE TRIP_ID = p_trip_id
    AND STATUS IN ('N', 'P');

    IF (p_occupied - p_current_no_tickets + p_no_tickets) > p_free_spots THEN
        RAISE_APPLICATION_ERROR(-20001, 'Not enough available spots on the trip...');
    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)
    SELECT RESERVATION_ID, SYSDATE, STATUS, p_no_tickets
    FROM RESERVATION
    WHERE RESERVATION_ID = p_reservation_id;

    COMMIT;
END;
```

Wywołania testowe

- Dodanie nowej rezerwacji:

```
BEGIN
    p_add_reservation(1, 3, 2);
END;
```

- Modyfikacja statusu rezerwacji:

```
BEGIN
    p_modify_reservation_status(27, 'P');
END;
```

- Modyfikacja ilości zamówionych biletów

```
BEGIN
    p_modify_reservation(41, 5);
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 - Sprawdzanie daty wycieczki

```
CREATE OR REPLACE TRIGGER T_CHECK_TRIP_DATE
BEFORE INSERT ON RESERVATION
FOR EACH ROW
DECLARE
    v_trip_date DATE;
BEGIN
    SELECT TRIP_DATE INTO v_trip_date
    FROM TRIP
    WHERE TRIP_ID = :NEW.TRIP_ID;

    IF v_trip_date < SYSDATE THEN
        RAISE_APPLICATION_ERROR(-20001, 'Picked an old trip...');
    END IF;
    COMMIT;
END;
```

Trigger - dostępność miejsc

```sql
CREATE OR REPLACE TRIGGER T_CHECK_FREESPOTS
BEFORE INSERT OR UPDATE ON RESERVATION
FOR EACH ROW
DECLARE
    v_max_places NUMBER;
    v_occupied NUMBER;
    v_current_tickets NUMBER;
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_occupied
    FROM RESERVATION
    WHERE TRIP_ID = :NEW.TRIP_ID
    AND STATUS IN ('N', 'P');

    v_current_tickets := COALESCE(:OLD.NO_TICKETS, 0);

    IF (v_occupied - v_current_tickets + :NEW.NO_TICKETS) > v_max_places THEN
        RAISE_APPLICATION_ERROR(-20002, 'Not enough available spots on the trip...');
    END IF;
    COMMIT;
END;
```

W ramach zadania powstały odpowiednie procedury, które nie posiadają sprawdzeń ilości miejsc oraz dat.

p_add_reservation_5

```
CREATE OR REPLACE PROCEDURE p_add_reservation_5 (
    p_trip_id NUMBER,
    p_person_id NUMBER,
    p_no_tickets NUMBER
) AS
    trip_date DATE;
    occupied NUMBER;
    free_spots NUMBER;
BEGIN
    BEGIN
        SELECT TRIP_DATE, MAX_NO_PLACES INTO trip_date, free_spots
        FROM TRIP WHERE TRIP.TRIP_ID = p_trip_id;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            RAISE_APPLICATION_ERROR(-20000, 'This trip does not exist...');
    END;

    INSERT INTO RESERVATION (TRIP_ID, PERSON_ID, STATUS, NO_TICKETS)
    VALUES (p_trip_id, p_person_id, 'N', p_no_tickets);

    COMMIT;
END;
```

p_modify_reservation_status_5

```
CREATE OR REPLACE PROCEDURE p_modify_reservation_status_5 (
    p_reservation_id NUMBER,
    p_status VARCHAR2
) AS
    current_status VARCHAR2(1);
    trip_id NUMBER;
    trip_date DATE;
    free_spots NUMBER;
    occupied NUMBER;
BEGIN
    BEGIN
        SELECT STATUS, TRIP_ID INTO current_status, trip_id
        FROM RESERVATION
        WHERE RESERVATION_ID = p_reservation_id;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            RAISE_APPLICATION_ERROR(-20000, 'Reservation not found...');
    END;

    UPDATE RESERVATION
    SET STATUS = p_status
    WHERE RESERVATION_ID = p_reservation_id;

    COMMIT;
END;
```

p_modify_reservation_5

```
CREATE OR REPLACE PROCEDURE p_modify_reservation_5 (
    p_reservation_id NUMBER,
    p_no_tickets NUMBER
) AS
    p_current_no_tickets NUMBER;
    p_trip_id NUMBER;
    p_free_spots NUMBER;
    p_occupied NUMBER;
BEGIN
    BEGIN
        SELECT NO_TICKETS, TRIP_ID INTO p_current_no_tickets, p_trip_id
        FROM RESERVATION
        WHERE RESERVATION_ID = p_reservation_id;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            RAISE_APPLICATION_ERROR(-20000, 'Reservation not found...');
    END;

    UPDATE RESERVATION
    SET NO_TICKETS = p_no_tickets
    WHERE RESERVATION_ID = p_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

```
CREATE OR REPLACE PROCEDURE p_recalculate_available_places AS
BEGIN
    FOR trip_rec IN (SELECT t.trip_id, t.max_no_places, COALESCE(SUM(r.no_tickets), 0) as occupied
                     FROM trip t
                     LEFT JOIN reservation r ON t.trip_id = r.trip_id AND r.status IN ('N', 'P')
                     GROUP BY t.trip_id, t.max_no_places)
    LOOP
        UPDATE trip
        SET no_available_places = trip_rec.max_no_places - trip_rec.occupied
        WHERE trip_id = trip_rec.trip_id;
    END LOOP;

    COMMIT;
END;
```

uruchomienie procedury:

```
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

Procedura dodająca rezerwację z aktualizacją no_available_places

```
CREATE OR REPLACE PROCEDURE p_add_reservation_6a (
    p_trip_id NUMBER,
    p_person_id NUMBER,
    p_no_tickets NUMBER
) AS
    trip_date DATE;
    available_places NUMBER;
BEGIN
    BEGIN
        SELECT TRIP_DATE, no_available_places
        INTO trip_date, available_places
        FROM TRIP
        WHERE TRIP.TRIP_ID = p_trip_id;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            RAISE_APPLICATION_ERROR(-10000, 'This trip does not exist...');
    END;

    IF trip_date < SYSDATE THEN
        RAISE_APPLICATION_ERROR(-10001, 'Cannot make reservation for past trip...');
    END IF;

    IF available_places < p_no_tickets THEN
        RAISE_APPLICATION_ERROR(-10002, 'Not enough available places on this trip...');
    END IF;

    INSERT INTO RESERVATION (TRIP_ID, PERSON_ID, STATUS, NO_TICKETS)
    VALUES (p_trip_id, p_person_id, 'N', p_no_tickets);

    UPDATE TRIP
    SET no_available_places = no_available_places - p_no_tickets
    WHERE TRIP_ID = p_trip_id;

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

Procedura modyfikująca status rezerwacji z aktualizacją no_available_places

```
CREATE OR REPLACE PROCEDURE p_modify_reservation_status_6a (
    p_reservation_id NUMBER,
    p_status VARCHAR2
) AS
    current_status VARCHAR2(1);
    trip_id NUMBER;
    no_tickets NUMBER;
    available_places NUMBER;
BEGIN
    BEGIN
        SELECT STATUS, TRIP_ID, NO_TICKETS
        INTO current_status, trip_id, no_tickets
        FROM RESERVATION
        WHERE RESERVATION_ID = p_reservation_id;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            RAISE_APPLICATION_ERROR(-10000, 'Reservation not found...');
    END;

    IF current_status = 'C' AND p_status IN ('N', 'P') THEN
        SELECT no_available_places INTO available_places
        FROM TRIP
        WHERE TRIP_ID = trip_id;

        IF available_places < no_tickets THEN
            RAISE_APPLICATION_ERROR(-10002, 'Not enough available places for reactivation...');
        END IF;

        UPDATE TRIP
        SET no_available_places = no_available_places - no_tickets
        WHERE TRIP_ID = trip_id;

    ELSIF current_status IN ('N', 'P') AND p_status = 'C' THEN
        UPDATE TRIP
        SET no_available_places = no_available_places + no_tickets
        WHERE TRIP_ID = trip_id;
    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, no_tickets);

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

Procedura modyfikująca liczbę biletów w rezerwacji

```
CREATE OR REPLACE PROCEDURE p_modify_reservation_6a (
    p_reservation_id NUMBER,
    p_no_tickets NUMBER
) AS
    current_no_tickets NUMBER;
    trip_id NUMBER;
    available_places NUMBER;
    current_status VARCHAR2(1);
BEGIN
    BEGIN
        SELECT NO_TICKETS, TRIP_ID, STATUS
        INTO current_no_tickets, trip_id, current_status
        FROM RESERVATION
        WHERE RESERVATION_ID = p_reservation_id;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            RAISE_APPLICATION_ERROR(-20000, 'Reservation not found...');
    END;

    IF current_status IN ('N', 'P') THEN
        SELECT no_available_places INTO available_places
        FROM TRIP
        WHERE TRIP_ID = trip_id;

        IF (p_no_tickets - current_no_tickets) > available_places THEN
            RAISE_APPLICATION_ERROR(-20001, 'Not enough available places...');
        END IF;

        UPDATE TRIP
        SET no_available_places = no_available_places - (p_no_tickets - current_no_tickets)
        WHERE TRIP_ID = trip_id;
    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, current_status, p_no_tickets);

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

Procedura modyfikująca maksymalną liczbę miejsc na wycieczce

```
CREATE OR REPLACE PROCEDURE p_modify_max_no_places_6a (
    p_trip_id NUMBER,
    p_max_no_places NUMBER
) AS
    current_max_places NUMBER;
    current_available_places NUMBER;
    reserved NUMBER;
BEGIN
    BEGIN
        SELECT MAX_NO_PLACES, no_available_places
        INTO current_max_places, current_available_places
        FROM TRIP
        WHERE TRIP_ID = p_trip_id;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            RAISE_APPLICATION_ERROR(-10000, 'Trip not found...');
    END;

    reserved := current_max_places - current_available_places;

    IF p_max_no_places < reserved THEN
        RAISE_APPLICATION_ERROR(-10000, 'Cannot reduce max places below reserved count...');
    END IF;

    UPDATE TRIP
    SET MAX_NO_PLACES = p_max_no_places,
        no_available_places = p_max_no_places - reserved
    WHERE TRIP_ID = p_trip_id;

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

Zaktualizowany widok dostępnych wycieczek

```
CREATE OR REPLACE VIEW vw_available_trip_6a AS
SELECT
    T.TRIP_ID, T.COUNTRY, T.TRIP_DATE, T.TRIP_NAME,
    T.MAX_NO_PLACES, T.no_available_places
FROM
    TRIP T
WHERE
    T.no_available_places > 0
    AND SYSDATE < T.TRIP_DATE;
```

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

Trigger aktualizujący no_available_places po dodaniu, zmianie lub usunięciu rezerwacji

```
CREATE OR REPLACE TRIGGER T_UPDATE_AVAILABLE_PLACES
AFTER INSERT OR UPDATE OR DELETE ON RESERVATION
FOR EACH ROW
DECLARE
    v_trip_id NUMBER;
    v_old_status CHAR(1);
    v_new_status CHAR(1);
    v_old_tickets NUMBER;
    v_new_tickets NUMBER;
    v_adjustment NUMBER := 0;
BEGIN
    IF INSERTING THEN
        v_trip_id := :NEW.TRIP_ID;
        v_new_status := :NEW.STATUS;
        v_new_tickets := :NEW.NO_TICKETS;
        IF v_new_status IN ('N', 'P') THEN
            v_adjustment := -v_new_tickets;
        END IF;

    ELSIF UPDATING THEN
        v_trip_id := :NEW.TRIP_ID;
        v_old_status := :OLD.STATUS;
        v_new_status := :NEW.STATUS;
        v_old_tickets := :OLD.NO_TICKETS;
        v_new_tickets := :NEW.NO_TICKETS;

        IF v_old_status = 'C' AND v_new_status IN ('N', 'P') THEN
            v_adjustment := -v_new_tickets;

        ELSIF v_old_status IN ('N', 'P') AND v_new_status = 'C' THEN
            v_adjustment := v_old_tickets;

        ELSIF v_old_status IN ('N', 'P') AND v_new_status IN ('N', 'P') THEN
            v_adjustment := v_old_tickets - v_new_tickets;
        END IF;

    ELSIF DELETING THEN
        v_trip_id := :OLD.TRIP_ID;
        v_old_status := :OLD.STATUS;
        v_old_tickets := :OLD.NO_TICKETS;

        IF v_old_status IN ('N', 'P') THEN
            v_adjustment := v_old_tickets;
        END IF;
    END IF;

    IF v_adjustment != 0 THEN
        UPDATE TRIP
        SET no_available_places = no_available_places + v_adjustment
        WHERE TRIP_ID = v_trip_id;
```

```
    END IF;
END;
```

Trigger sprawdzający dostępność miejsc przed dodaniem lub aktualizacją rezerwacji

```
CREATE OR REPLACE TRIGGER T_CHECK_AVAILABLE_PLACES
BEFORE INSERT OR UPDATE ON RESERVATION
FOR EACH ROW
DECLARE
    v_available_places NUMBER;
    v_trip_date DATE;
BEGIN
    SELECT no_available_places, TRIP_DATE
    INTO v_available_places, v_trip_date
    FROM TRIP
    WHERE TRIP_ID = :NEW.TRIP_ID;

    IF v_trip_date < SYSDATE AND INSERTING THEN
        RAISE_APPLICATION_ERROR(-20001, 'Cannot make reservation for past trip...');
    END IF;

    IF INSERTING AND :NEW.STATUS IN ('N', 'P') THEN
        IF v_available_places < :NEW.NO_TICKETS THEN
            RAISE_APPLICATION_ERROR(-20002, 'Not enough available places...');
        END IF;

    ELSIF UPDATING THEN
        IF :OLD.STATUS = 'C' AND :NEW.STATUS IN ('N', 'P') THEN
            IF v_available_places < :NEW.NO_TICKETS THEN
                RAISE_APPLICATION_ERROR(-20003, 'Not enough available places for reactivation...');
            END IF;
        ELSIF :OLD.STATUS IN ('N', 'P') AND :NEW.STATUS IN ('N', 'P') AND :OLD.NO_TICKETS != :NEW.NO_TICKETS THEN
            IF v_available_places < (:NEW.NO_TICKETS - :OLD.NO_TICKETS) THEN
                RAISE_APPLICATION_ERROR(-20004, 'Not enough available places for ticket increase...');
            END IF;
        END IF;
    END IF;
END;
```

Trigger aktualizujący no_available_places po zmianie max_no_places

```
CREATE OR REPLACE TRIGGER T_UPDATE_PLACES_ON_MAX_CHANGE
BEFORE UPDATE OF MAX_NO_PLACES ON TRIP
FOR EACH ROW
DECLARE
    v_reserved NUMBER;
BEGIN
    v_reserved := :OLD.MAX_NO_PLACES - :OLD.no_available_places;
    IF :NEW.MAX_NO_PLACES < v_reserved THEN
        RAISE_APPLICATION_ERROR(-20005, 'Cannot reduce max places below reserved count...');
    END IF;

    :NEW.no_available_places := :NEW.MAX_NO_PLACES - v_reserved;
END;
```

Procedura dodająca rezerwację

```
CREATE OR REPLACE PROCEDURE p_add_reservation_6b (
    p_trip_id NUMBER,
    p_person_id NUMBER,
    p_no_tickets NUMBER
) AS
BEGIN
    INSERT INTO RESERVATION (TRIP_ID, PERSON_ID, STATUS, NO_TICKETS)
    VALUES (p_trip_id, p_person_id, 'N', p_no_tickets);

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

Procedura modyfikująca status rezerwacji

```
CREATE OR REPLACE PROCEDURE p_modify_reservation_status_6b (
    p_reservation_id NUMBER,
    p_status VARCHAR2
) AS
BEGIN
    UPDATE RESERVATION
    SET STATUS = p_status
    WHERE RESERVATION_ID = p_reservation_id;

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

Procedura modyfikująca liczbę biletów w rezerwacji

```
CREATE OR REPLACE PROCEDURE p_modify_reservation_6b (
    p_reservation_id NUMBER,
    p_no_tickets NUMBER
) AS
BEGIN
    UPDATE RESERVATION
    SET NO_TICKETS = p_no_tickets
    WHERE RESERVATION_ID = p_reservation_id;

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

Procedura modyfikująca maksymalną liczbę miejsc na wycieczce

```
CREATE OR REPLACE PROCEDURE p_modify_max_no_places_6b (
    p_trip_id NUMBER,
    p_max_no_places NUMBER
) AS
BEGIN
    UPDATE TRIP
    SET MAX_NO_PLACES = p_max_no_places
    WHERE TRIP_ID = p_trip_id;

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

Zaktualizowany widok dostępnych wycieczek

```
CREATE OR REPLACE VIEW vw_available_trip_6b AS
SELECT
    T.TRIP_ID, T.COUNTRY, T.TRIP_DATE, T.TRIP_NAME,
    T.MAX_NO_PLACES, T.no_available_places
FROM
    TRIP T
WHERE
    T.no_available_places > 0
    AND SYSDATE < T.TRIP_DATE;
```

Zadanie 7 - podsumowanie

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

Porównanie Oracle PL/SQL i Microsoft SQL Server T-SQL

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

Podobieństwa:

- **Proceduralność** – Obie technologie umożliwiają tworzenie procedur składowanych, funkcji, wyzwalaczy i pakietów, co pozwala na bardziej zaawansowaną logikę w bazie danych.
- **Transakcyjność** – Zarówno PL/SQL, jak i T-SQL obsługują transakcje, co pozwala na grupowanie wielu operacji na bazie danych w jedną jednostkę pracy.
- **Obsługa błędów** – Obie technologie umożliwiają obsługę błędów, choć w inny sposób: w T-SQL używa się `TRY...CATCH`, a w PL/SQL `BEGIN...EXCEPTION...END`.

Różnice:

- **Składnia** – Składnia w PL/SQL i T-SQL różni się w pewnych aspektach, np. w PL/SQL do łączenia ciągów znaków używa się `||`, a w T-SQL `+`.
- **Obsługa NULL** – W T-SQL porównanie wartości z `NULL` zwraca `NULL`, natomiast w PL/SQL porównanie `NULL` z `NULL` zwraca `TRUE`.
- **Funkcje wbudowane** – Oba języki mają różne funkcje wbudowane, np. T-SQL używa `GETDATE()` do pobierania bieżącej daty i godziny, a PL/SQL `SYSDATE`.
- **Obsługa bloków kodu** – W PL/SQL bloki kodu definiuje się za pomocą `DECLARE`, `BEGIN`, `EXCEPTION` i `END`, podczas gdy w T-SQL struktura kodu jest prostsza i często korzysta się z `BEGIN...END`.