Baza danych dla biura turystycznego

Autorzy: Czernecki Paweł, Dziarkowski Michał, Matuszyński Wojciech, Szkarłat Szymon

1. Wymagania i funkcje systemu

1.1. Role:

W bazie danych mogą istnieć podmioty, które mają więcej niż jedną rolę. Jeżeli podmiot jest zarówno uczestnikiem jak i klientem wycieczki, zyskuje uprawnienia związane z obiema rolami. W przypadku, gdy podmiot jest jednocześnie klientem biura i jego administratorem, przewidziane są dwa osobne konta (jedno na stronie biura, drugie w panelu administracyjnym).

1.1.1. Właściciel biura

- Posiada minimalne uprawnienia w systemie.
- Jego uprawnienia nie przekraczają uprawnienia zwykłego użytkownika systemu

1.1.2. Admin

- Posiada konto w panelu administracyjnym.
- Może tworzyć/usuwać nowe wycieczki i zmieniać ich właściwości, w wypadku uzyskania odpowiedniego komunikatu od sekretarza/sekretarki.
- Ma dostęp do listy uczestników i listy klientów dla każdej wycieczki/fakultetu. Może usunąć klienta z listy klientów wycieczki, jeżeli nie dopełni swych obowiązków.

1.1.3. Sekretarz/Sekretarka

- Odpowiada za płatności przebiegające między klientami a systemem.
- W przypadku rezygnacji z wycieczki/fakultetu bądź niespełnienia wymogów, zwraca pieniądze klientom na ich konto.
- Kontaktuje się z firmami przewoźniczymi i hotelarskimi w celu uzgodnienia zakwaterowania uczestników, a także dat i miejsc zbiórek.
- Tydzień przed rozpoczęciem wycieczki wysyła administratorom listę klientów, którzy dopełnili obowiązków.
- Informuje administratorów na temat zmiany/dodania szczegółów wycieczki (np. zmiana daty rozpoczęcia, udostępnienie miejscu i czasu zbiórki na wycieczkę).

1.1.4. Klient

- Może być podmiotem indywidualnym lub firmą.
- Posiada konto na stronie.
- Może rezerwować wycieczki/fakultety.
- Może rezygnować z wycieczki/fakultetu.
- Może podawać dane uczestników.
- Ma dostęp do listy zarezerwowanych wycieczek oraz ma dostęp do listy uczestników, uczestniczących w zarezerwowanej wycieczce.

1.1.5. Uczestnik

- Posiada konto na stronie.
- Może zobaczyć wycieczki i fakultety, na które jest zapisany. Dla każdej wycieczki/fakultetu może sprawdzić miejsce i czas zbiórki(np. czas wylotu i lokalizacja lotniska na wycieczkę zagranicą), a dla wycieczek może też sprawdzić miejsce i czas przybycia na miejsce, oraz datę powrotu.

1.2. Działanie biura turystycznego

1.2.1. Wycieczki

- Główna część oferty biura podróży
- Składa się z jedno- i wielodniowych wypraw, zakup wycieczki wiąże się z przygotowaniem dojazdu i zakwaterowania dla uczestników takowej oferty.
- Zakup atrakcji/fakultetu jest możliwy tylko w przypadku uprzednio zakupionej wycieczki.
- Każda wycieczka ma własny dla siebie limit miejsc oraz cenę za osobę.

1.2.2. Fakultety (Atrakcje dodatkowe)

- Dodatkowa/opcjonalna część oferty biura.
- Klient może urozmacić wycieczkę uczestników poprzez zakup usług dodatkowych, które odbędą się w czasie wycieczki.
- Aby uczestnik mógł wziąć udział w fakultecie, musi być uczestnikiem związanej wycieczki.
- Każdy fakultet ma własny dla siebie limit miejsc oraz dodatkową cenę za osobę.

1.2.3. Rezerwacje Ofert

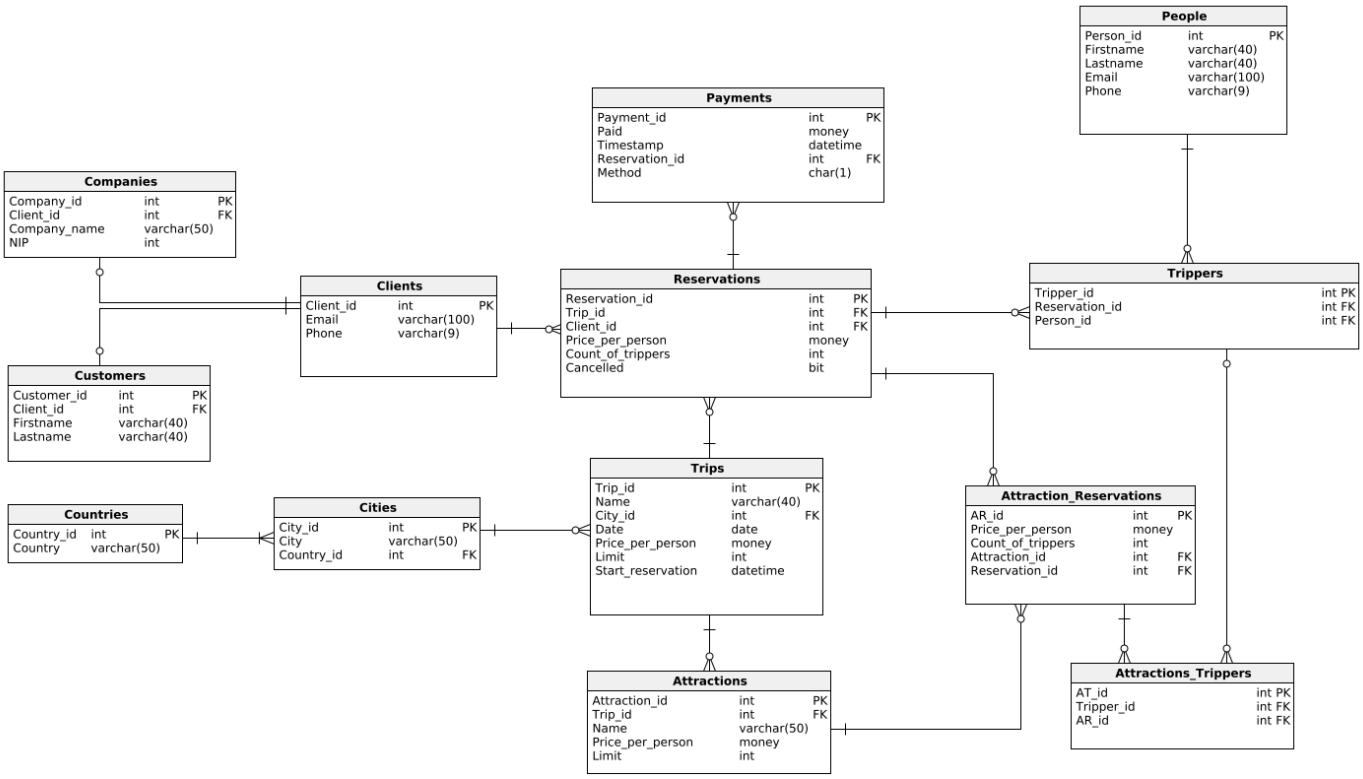
- Klienci rezerwują wycieczki i fakultety w imieniu jednego lub wielu uczestników.
- Przy rezerwacji wycieczki podają liczbę uczestników.
- W terminie do tygodnia przed rozpoczęciem wycieczki klienci są zobowiązani do podania listy uczestników (mowa tutaj o podaniu danych kontaktowych uczestników: imię, nazwisko, email oraz nr telefonu), oraz wpłacenia pełnej kwoty za wycieczkę.
- W przypadku niedopełnienia obowiązku, rezerwacja jest usuwana przez administratora, a wpłacone dotychczas pieniądze zwracane są przez sekretarkę.
- Fakultet można zamówić w dowolnym momencie po zamówieniu wycieczki, do tygodnia przed datą rozpoczęcia wycieczki. Musi przypisać do niego uczestnika imiennie do tygodnia przed rozpoczęciem wycieczki.
- W przypadku, kiedy klient zmniejszy liczbę uczestników biorących udział w wycieczce/fakultecie, przysługuje mu zwrot wpłaconych pieniędzy pod warunkiem, że poinformował o tym fakcie biuro na tydzień przed rozpoczęciem wycieczki.

1.2.4. Zbiórki

- Informacje na temat lokalizacji i daty, gdzie mają zebrać się uczestnicy w celu rozpoczęcia wycieczki dostarczane są nie później niż tydzień przed rozpoczęciem wycieczki.
- Data zbiórki wycieczki może być jednorazowo przesunięta do 24 godzin w przyszłość, nie później niż dobę przed oryginalną datą zbiórki.
- Informacje na temat zbiórki na usługi fakultatywne pojawiają się nie później niż dobę przed datą rozpoczęcia usługi fakultatywnej.

2. Baza danych

2.1. Schemat bazy danych



2.2. Opis poszczególnych tabel

Clients

Tabela zawiera klientów, którzy złożyli zamówienia na dane wycieczki, dokonali rezerwacji dla określonej liczby osób. Mogą to być zarówno osoby prywatne, jak i firmy. Dodatkowo mamy sprawdzenie czy stan konta jest dodatni.

Nazwa atrybutu	Typ	Opis/Uwagi
Client_id	int	Primary Key (PK)
Email	varchar(100)	Adres email
Phone	varchar(9)	Nr telefonu

```
CREATE TABLE Clients (
    Client_id int  NOT NULL IDENTITY(1, 1),
    Email varchar(100)  NOT NULL,
    Phone varchar(9)  NOT NULL,
    CONSTRAINT Clients_pk PRIMARY KEY  (Client_id)
);
```

Companies

Tabela zawiera firmy, które dokonały rezerwacji wycieczki w systemie oraz dane je identyfikujące, tj. nazwa firmy oraz nr NIP.

Nazwa atrybutu	Typ	Opis/Uwagi
Company_id	int	PK
Client_id	int	FK
Company_name	varchar(50)	Nazwa firmy
NIP	int	Nr NIP

```
CREATE TABLE Companies (
    Company_id int  NOT NULL IDENTITY(1, 1),
    Client_id int  NOT NULL,
    Company_name varchar(50)  NOT NULL,
    NIP int  NOT NULL,
    CONSTRAINT Companies_pk PRIMARY KEY  (Company_id)
);
```

Definiowanie relacji pomiędzy tabelą Companies a Clients.

```
ALTER TABLE Companies ADD CONSTRAINT Companies_Clients
    FOREIGN KEY (Client_id)
    REFERENCES Clients (Client_id);
```

Customers

Tabela zawiera informacje o klienta (osobach prywatnych), którzy dokonali rezerwacji wycieczki w systemie.

Nazwa atrybutu	Typ	Opis/Uwagi
Customer_id	int	PK
Client_id	int	FK
Firstname	varchar(40)	Imię
Lastname	varchar(40)	Nazwisko

```
CREATE TABLE Customers (
    Customer_id int  NOT NULL IDENTITY(1, 1),
    Client_id int  NOT NULL,
    Firstname varchar(40)  NOT NULL,
    Lastname varchar(40)  NOT NULL,
    CONSTRAINT Customers_pk PRIMARY KEY  (Customer_id)
);
```

Definiowanie relacji pomiędzy tabelą Customers a Clients.

```
ALTER TABLE Customers ADD CONSTRAINT Customers_Clients
    FOREIGN KEY (Client_id)
    REFERENCES Clients (Client_id);
```

Trips

Tabela zawiera wycieczki, które oferuje biuro turystyczne. Dodatkowo mamy sprawdzenie czy limit oraz cena za osobę są wartościami dodatnimi.

Nazwa atrybutu	Typ	Opis/Uwagi
Trip_id	int	PK
Name	varchar(40)	Nazwa wycieczki
City_id	int	FK
Date	date	Data odbycia się wycieczki
Price_per_person	money	Cena za osobę
Limit	int	Limit osób na wycieczce
Start_reservation	datetime	Data rozpoczęcia rezerwacji

```
CREATE TABLE Trips (
    Trip_id int  NOT NULL IDENTITY(1, 1),
    Name varchar(40)  NOT NULL,
    City_id int  NOT NULL,
    Date date  NOT NULL,
    Price_per_person money  NOT NULL,
    Limit int  NOT NULL,
    Start_reservation datetime  NOT NULL,
    CONSTRAINT check_limit CHECK (Limit > 0),
    CONSTRAINT check_price_per_person CHECK (Price_per_person > 0),
    CONSTRAINT Trips_pk PRIMARY KEY  (Trip_id)
);
```

Definiowanie relacji pomiędzy tabelą Trips a Cities.

```
ALTER TABLE Trips ADD CONSTRAINT Trips_Cities
    FOREIGN KEY (City_id)
    REFERENCES Cities (City_id);
```

Cities

Tabela zawiera nazwy miast, w których odbywają się wycieczki.

Nazwa atrybutu	Typ	Opis/Uwagi
City_id	int	PK
City	varchar(50)	Miasto

Nazwa atrybutu	Typ	Opis/Uwagi
Country_id	int	FK

```
CREATE TABLE Cities (
    City_id int  NOT NULL IDENTITY(1, 1),
    City varchar(50)  NOT NULL,
    Country_id int  NOT NULL,
    CONSTRAINT Cities_pk PRIMARY KEY  (City_id)
);
```

Definiowanie relacji pomiędzy tabelą Cities a Countries

```
ALTER TABLE Cities ADD CONSTRAINT Cities_Countries
    FOREIGN KEY (Country_id)
    REFERENCES Countries (Country_id);
```

Countries

Tabela słownikowa zawierająca dopuszczalne nazwy państw.

Nazwa atrybutu	Typ	Opis/Uwagi
Customer_id	int	PK
Country	varchar(50)	Nazwa kraju

```
CREATE TABLE Countries (
    Country_id int  NOT NULL IDENTITY(1, 1),
    Country varchar(50)  NOT NULL,
    CONSTRAINT Countries_pk PRIMARY KEY  (Country_id)
);
```

Attractions

Atrakcje oferowane przez biuro podróży. Aby można było dokonać zakupu atrakcji dodatkowych należy najpierw zakupić wycieczkę. Dodatkowo mamy sprawdzenie czy limit miejsc oraz cena za osobę są dodatnie.

Nazwa atrybutu	Typ	Opis/Uwagi
Attraction_id	int	PK
Trip_id	int	FK
Name	varchar(50)	Nazwa atrakcji
Price_per_person	money	Cena za osobę
Limit	int	Limit osób na wycieczce

```
CREATE TABLE Attractions (
    Attraction_id int  NOT NULL IDENTITY(1, 1),
    Trip_id int  NOT NULL,
    Name varchar(50)  NOT NULL,
    Price_per_person money  NOT NULL,
    Limit int  NOT NULL,
    CONSTRAINT Attractions_check_limit CHECK (Limit > 0),
    CONSTRAINT Attractions_check_price_per_person CHECK (Price_per_person > 0),
    CONSTRAINT Attractions_pk PRIMARY KEY  (Attraction_id)
);
```

Definiowanie relacji pomiędzy tabelą Attractions a Trips_Attractions

```
ALTER TABLE Attractions ADD CONSTRAINT Trips_Attractions
    FOREIGN KEY (Trip_id)
    REFERENCES Trips (Trip_id);
```

Attraction_Reservations

W tej tabeli umieszczane są zarezerwowane atrakcje. Dodatkowo mamy sprawdzenie czy liczba zarezerwowanych miejsc jest dodatnia oraz czy cena za osobę również jest dodatnia.

Nazwa atrybutu	Typ	Opis/Uwagi
AR_id	int	Rezerwacja atrakcji (PK)
Price_per_person	money	Cena za osobę
Count_of_trippers	int	Liczba zarezerwowanych miejsc
Attraction_id	in	FK

Nazwa atrybutu	Typ	Opis/Uwagi
Reservation_id	int	FK

```
CREATE TABLE Attraction_Reservations (
    AR_id int  NOT NULL IDENTITY(1, 1),
    Price_per_person money  NOT NULL,
    Count_of_trippers int  NOT NULL,
    Attraction_id int  NOT NULL,
    Reservation_id int  NOT NULL,
    CONSTRAINT Attraction_Reservations_check_count_of_trippers CHECK (Count_of_trippers > 0),
    CONSTRAINT Attraction_Reservations_check_price_per_person CHECK (Price_per_person > 0),
    CONSTRAINT Attraction_Reservations_pk PRIMARY KEY  (AR_id)
);
```

Definiowanie relacji pomiędzy tabelą Attraction_Reservations a Attractions.

```
ALTER TABLE Attraction_Reservations ADD CONSTRAINT Attraction_Reservations_Attractions
    FOREIGN KEY (Attraction_id)
    REFERENCES Attractions (Attraction_id);
```

Definiowanie relacji pomiędzy tabelą Attraction_Reservations a Reservations.

```
ALTER TABLE Attraction_Reservations ADD CONSTRAINT Attraction_Reservations_Reservations
    FOREIGN KEY (Reservation_id)
    REFERENCES Reservations (Reservation_id);
```

Attractions_Trippers

Tabela zawiera wylistowanych uczestników poszczególnych atrakcji.

Nazwa atrybutu	Typ	Opis/Uwagi
AT_id	int	Atracja dla danej wycieczki (PK)
Tripper_id	int	FK
AR_id	int	FK

```
CREATE TABLE Attractions_Trippers (
    AT_id int  NOT NULL IDENTITY(1, 1),
    Tripper_id int  NOT NULL,
    AR_id int  NOT NULL,
    CONSTRAINT Attractions_Trippers_pk PRIMARY KEY  (AT_id)
);
```

Definiowanie relacji pomiędzy tabelą Attractions_Trippers a Attraction_Reservations.

```
ALTER TABLE Attractions_Trippers ADD CONSTRAINT Attractions_Trippers_Attraction_Reservations
    FOREIGN KEY (AR_id)
    REFERENCES Attraction_Reservations (AR_id);
```

Definiowanie relacji pomiędzy tabelą Attractions_Trippers a Trippers.

```
ALTER TABLE Attractions_Trippers ADD CONSTRAINT Attractions_Trippers_Trippers
    FOREIGN KEY (Tripper_id)
    REFERENCES Trippers (Tripper_id);
```

Reservations

W tabeli tej mamy umieszczone informacje o wszystkich dokonanych rezerwacjach. Dodano również sprawdzenie czy cena za osobę oraz liczba zarezerwowanych miejsc są dodatnie.

Nazwa atrybutu	Typ	Opis/Uwagi
Reservation_id	int	PK
Trip_id	int	FK
Client_id	int	FK
Count_of_trippers	int	Liczba zarezerwowanych miejsc
Cancelled	bit	Informacja czy rezerwacja została anulowana

```
CREATE TABLE Reservations (
    Reservation_id int  NOT NULL IDENTITY(1, 1),
```

```
    Trip_id int  NOT NULL,
    Client_id int  NOT NULL,
    Price_per_person money  NOT NULL,
    Count_of_trippers int  NOT NULL,
    Cancelled bit  NOT NULL,
    CONSTRAINT check_price_per_person CHECK (Price_per_person > 0),
    CONSTRAINT Reservations_pk PRIMARY KEY  (Reservation_id)
);
```

Definiowanie relacji pomiędzy tabelą Reservations a Clients.

```
ALTER TABLE Reservations ADD CONSTRAINT Reservations_Clients
    FOREIGN KEY (Client_id)
    REFERENCES Clients (Client_id);
```

Definiowanie relacji pomiędzy tabelą Reservations a Trips.

```
ALTER TABLE Reservations ADD CONSTRAINT Reservations_Trips
    FOREIGN KEY (Trip_id)
    REFERENCES Trips (Trip_id);
```

Trippers

Tabela zawiera wylistowanych uczestników poszczególnych wycieczek, które znajdują się w ofercie biura turystycznego.

Nazwa atrybutu	Typ	Opis/Uwagi
Tripper_id	int	PK
Reservation_id	int	FK
Person_id	int	FK

```
CREATE TABLE Trippers (
    Tripper_id int  NOT NULL IDENTITY(1, 1),
    Reservation_id int  NOT NULL,
    Person_id int  NOT NULL,
    CONSTRAINT Trippers_pk PRIMARY KEY  (Tripper_id)
);
```

Definiowanie relacji pomiędzy tabelą Trippers a People.

```
ALTER TABLE Trippers ADD CONSTRAINT Trippers_People
    FOREIGN KEY (Person_id)
    REFERENCES People (Person_id);
```

Definiowanie relacji pomiędzy tabelą Trippers a Reservations.

```
ALTER TABLE Trippers ADD CONSTRAINT Trippers_Reservations
    FOREIGN KEY (Reservation_id)
    REFERENCES Reservations (Reservation_id);
```

People

Tabela zawiera dane uczestników poszczególnych wycieczek.

Nazwa atrybutu	Typ	Opis/Uwagi
Person_id	int	PK
Firstname	varchar(40)	Imię
Lastname	varchar(40)	Nazwisko
Email	varchar(100)	Adres email
Phone	varchar(9)	Nr telefonu

```
CREATE TABLE People (
    Person_id int  NOT NULL IDENTITY(1, 1),
    Firstname varchar(40)  NOT NULL,
    Lastname varchar(40)  NOT NULL,
    Email varchar(100)  NOT NULL,
    Phone varchar(9)  NOT NULL,
    CONSTRAINT People_pk PRIMARY KEY  (Person_id)
);
```

Payments

Tabela zawiera wszystkie operacje (wpłaty i zwrot na konto) jakich dokonano w biurze turystycznym. Dodano sprawdzenie czy kwota transakcji jest dodatnia.

Nazwa atrybutu	Typ	Opis/Uwagi
Payment_id	int	PK
Paid	money	Kwota zapłacona/wypłacona
Timestamp	datetime	Czas dokonania transakcji
Reservation_id	int	FK
Method	char(1)	Metoda płatności

```
CREATE TABLE Payments (
    Payment_id int  NOT NULL IDENTITY(1, 1),
    Paid money  NOT NULL,
    Timestamp datetime  NOT NULL,
    Reservation_id int  NOT NULL,
    Method char(1)  NOT NULL,
    CONSTRAINT Payments_check_paid CHECK (Paid > 0),
    CONSTRAINT Payments_check_method CHECK (Method IN ('M', 'C', 'P', 'B')),
    CONSTRAINT Payments_pk PRIMARY KEY  (Payment_id)
);
```

Definiowanie relacji pomiędzy tabelą Payments a Reservations.

```
ALTER TABLE Payments ADD CONSTRAINT Payments_Reservations
    FOREIGN KEY (Reservation_id)
    REFERENCES Reservations (Reservation_id);
```

3. Widoki, procedury/funkcje, triggery

Widoki

Trip_Offers

Oferty wycieczek w biurze turystycznym.

```
create or alter view Trip_Offers as
select
    t.Trip_id,
    t.Name,
    ci.City,
    co.Country,
    t.Price_per_person,
    t.Limit,
    t.Date
from Trips t
join Cities ci on ci.City_id=t.City_id
join Countries co on co.Country_id=ci.Country_id;
```

Trip_id	Name	City	Country	Price_per_person	Limit	Date
1	Discover Paris	Paris	France	500,00	20	2024-06-15
2	Explore Rome	Rome	Italy	600,00	15	2024-07-10
3	Adventures in Tokyo	Tokyo	Japan	700,00	25	2024-08-20
4	Safari in Guadalajara	Guadalajara	Mexico	650,00	18	2024-09-05
5	Cruise in Sydney	Sydney	Australia	900,00	30	2024-10-15
6	Hike in Machu Picchu	Lima	Peru	1080,00	22	2024-11-10
7	Holidays in USA	New York	United States	5100,00	17	2024-12-01
8	Relaxing in Athens	Athens	Greece	1200,00	28	2025-01-20
9	Snorkelling in Australia	Sydney	Australia	4500,00	10	2024-06-21

Trips_value_for_clients

Widok, który dla każdego klienta pokazuje ile będą go kosztować zarezerwowane przez niego wycieczki i atrakcje. *left join* został użyty po to, aby uwzględnić również nieopłacone jeszcze rezerwacje - takie, dla których nie dokonano jeszcze wpłat (brak wpisów w tabeli Payments).

```
create view Trips_value_for_clients as
select
    c.Client_id,
    COALESCE((r.Count_of_trippers * r.Price_per_person) +
```

```
    (at.Count_of_trippers * at.Price_per_person),0)
    Trips_and_attractions_value
from Reservations r
left join Clients c on c.Client_id=r.Client_id
left join Attraction_Reservations at on at.Reservation_id=r.Reservation_id
where Cancelled = 'false';
```

Client_id	Trips_and_attractions_value
15	540,00
5	0,00
18	0,00
20	6960,00

Trippers_lists

Lista imiennych uczestników na poszczególne wycieczki, które oferuje biuro turystyczne.

```
create view Trippers_lists as
select
    Trips.Trip_id,
    Trips.Name TripName,
    p.Firstname,
    p.Lastname
from Reservations r
join Trippers on Trippers.Reservation_id=r.Reservation_id
join People p on p.Person_id=Trippers.Person_id
join Trips on Trips.Trip_id=r.Trip_id;
```

TripName	Firstname	Lastname
Discover Paris	Mateusz	Kwiatkowska
Adventures in Tokyo	Jan	Nowak
Adventures in Tokyo	Marta	Kowalska
Adventures in Tokyo	Tadeusz	Wisniewski
Adventures in Tokyo	Katarzyna	Wójcik
Hike in Machu Picchu	Jan	Dabrowski
Hike in Machu Picchu	Aleksandra	Kozlowska
Hike in Machu Picchu	Pawel	Jankowski
Hike in Machu Picchu	Weronika	Wojciechowska
Hike in Machu Picchu	Mateusz	Kwiatkowska

Available_Trips

Widok wypisujący informacje dla klientów na temat wycieczek, które są wciąż dostępne (są na nie wolne miejsca, a sama wycieczka jeszcze się nie odbyła).

```
create or alter view Available_Trips as
select
    t.Trip_id,
    t.Name as TripName,
    ci.City as CityName,
    co.Country as CountryName,
    t.Date as TripDate,
    t.Price_per_person,
    t.Limit,
    t.Limit - ISNULL(SUM(r.Count_of_trippers), 0) as AvailableSpots
from Trips t
left join Cities ci on ci.City_id = t.City_id
left join Countries co on co.Country_id = ci.Country_id
left join Reservations r on r.Trip_id = t.Trip_id and r.Cancelled = 0
where t.Date > getdate()
group by t.Trip_id, t.Name, ci.City, co.Country, t.Date, t.Price_per_person, t.Limit
having t.Limit - ISNULL(SUM(r.Count_of_trippers), 0) != 0;
```

Trip_id	TripName	CityName	CountryName	TripDate	Price_per_person	Limit	AvailableSpots
1	Discover Paris	Paris	France	2024-06-15	500,00	20	19
2	Explore Rome	Rome	Italy	2024-07-10	600,00	15	15
3	Adventures in Tokyo	Tokyo	Japan	2024-08-20	700,00	25	21
4	Safari in Guadalajara	Guadalajara	Mexico	2024-09-05	650,00	18	14
5	Cruise in Sydney	Sydney	Australia	2024-10-15	900,00	30	30

Trip_id	TripName	CityName	CountryName	TripDate	Price_per_person	Limit	AvailableSpots
6	Hike in Machu Picchu	Lima	Peru	2024-11-10	1080,00	22	16
7	Holidays in USA	New York	United States	2024-12-01	5100,00	17	17
8	Relaxing in Athens	Athens	Greece	2025-01-20	1200,00	28	28
9	Snorkelling in Australia	Sydney	Australia	2024-06-21	4500,00	10	10

Widok w celach archiwalnych, pokazujący wszystkie dotychczasowe płatności

```
create view Payment_Details_History as
select
    pa.payment_id,
    pa.paid,
    pa.timestamp,
    pa.method,
    r.reservation_id,
    c.client_id,
    case
        when cust.customer_id is not null then cust.firstname + ' ' + cust.lastname
        when comp.company_id is not null then comp.company_name
    end as clientname,
    t.trip_id,
    t.name as tripname
from payments pa
join reservations r on r.reservation_id = pa.reservation_id
join clients c on c.client_id = r.client_id
left join customers cust on cust.client_id = c.client_id
left join companies comp on comp.client_id = c.client_id
left join trips t on t.trip_id = r.trip_id;
```

payment_id	paid	timestamp	method	reservation_id	client_id	clientname	trip_id	tripname
19	540,00	2024-05-19 18:31:34.000	M	1	15	Andrzej Kowalczyk	1	Discover Paris
20	3120,00	2024-05-19 18:31:34.000	M	2	5	Future Builders	3	Adventures in Tokyo
21	2600,00	2024-05-19 18:31:34.000	C	4	18	Joanna Zielinska	4	Safari in Guadalajara
22	7200,00	2024-05-19 18:31:34.000	P	5	20	Magdalena Wozniak	6	Hike in Machu Picchu

Client_List

Lista klientów wraz z listą wykonanych przez nich rezerwacji (łącznie z anulowanymi)

```
create or alter view Client_List as
select C.Client_id, 'FIRMA' as Client_Type,  Cm.Company_name as Nazwa, C.Phone, C.Email, Cm.NIP,
       count(R2.Reservation_id) as Liczba_rezerwacji
from Companies Cm
join dbo.Clients C on Cm.Client_id = C.Client_id
left join dbo.Reservations R2 on C.Client_id = R2.Client_id
group by C.Client_id, Cm.Company_name, C.Phone, C.Email, Cm.NIP
union
select C.Client_id, 'OSOBA FIZYCZNA' as Client_Type, P.Firstname + ' ' + P.Lastname, C.Phone, C.Email, null,
       count(R3.Reservation_id) as Liczba_rezerwacji
from Customers P
join dbo.Clients C on P.Client_id = C.Client_id
left join dbo.Reservations R3 on C.Client_id = R3.Client_id
group by C.Client_id, P.Firstname, P.Lastname, C.Phone, C.Email
```

Client_id	Client_Type	Nazwa	Phone	Email	NIP	Liczba_rezerwacji
1	FIRMA	Tech Innovators	123456789	techinnovators@example.com	1234567890	0
2	FIRMA	Green Solutions	987654321	greensolutions@greentech.net	1234567891	0
3	FIRMA	Health First	456789123	info@healthfirst.net	1234567892	0
4	FIRMA	Creative Minds	789123456	support@creativeminds.tech	1234567893	0
5	FIRMA	Future Builders	321654987	builders@thefuturebuildersgroup.com	1234567894	1
6	FIRMA	Smart Home Inc.	654987321	contact@futurebuilders.net	1234567895	0
7	FIRMA	Digital Dynamics	147258369	contact@digitaldynamics.org	1234567896	0
8	FIRMA	Eco Energy	258369147	support@ecoenergy.org	1234567897	0
9	FIRMA	Finance Hub	369147258	support@financehub.org	1234567898	0
10	FIRMA	Fashion Forward	963852741	sales@fashionforward.tech	1234567899	0
11	OSOBA FIZYCZNA	Jan Kowalski	741852963	kowalski@example.com		0
12	OSOBA FIZYCZNA	Anna Nowak	852963147	nowak@example.com		0

Client_id	Client_Type	Nazwa	Phone	Email	NIP	Liczba_rezerwacji
13	OSOBA FIZYCZNA	Piotr Wisniewski	369852147	wisniewski@example.com		1
14	OSOBA FIZYCZNA	Katarzyna Wójcik	258741369	wojcik@example.com		0
15	OSOBA FIZYCZNA	Andrzej Kowalczyk	123789456	kowalczyk@example.com		1
16	OSOBA FIZYCZNA	Malgorzata Kaminska	987654123	kaminska@example.com		0
17	OSOBA FIZYCZNA	Stanislaw Lewandowski	654123789	lewandowski@example.com		0
18	OSOBA FIZYCZNA	Joanna Zielinska	456321987	zielinska@example.com		1
19	OSOBA FIZYCZNA	Tomasz Szymanski	789456123	szymanski@example.com		0
20	OSOBA FIZYCZNA	Magdalena Wozniak	321987654	wozniak@example.com		1

Cities_List

Widok wyświetlający wszystkie miasta z naszego systemu wraz z krajami.

```
create or alter view City_List as
select distinct Cities.City_id , Cities.City, C.Country
from Cities
join dbo.Countries C on C.Country_id = Cities.Country_id
order by C.Country, Cities.City
```

City_id	City	Country
18	Brisbane	Australia
17	Melbourne	Australia
16	Sydney	Australia
6	Lyon	France
5	Nice	France
4	Paris	France
19	Athens	Greece
21	Mykonos	Greece
20	Santorini	Greece
3	Florence	Italy
1	Rome	Italy
2	Venice	Italy
14	Kyoto	Japan
15	Osaka	Japan
13	Tokyo	Japan
22	Cancun	Mexico
24	Guadalajara	Mexico
23	Mexico City	Mexico
28	Cuzco	Peru
30	Jaen	Peru
29	Lima	Peru
7	Barcelona	Spain
8	Madrid	Spain
9	Seville	Spain
25	Bangkok	Thailand
26	Chiang Mai	Thailand
27	Phuket	Thailand
12	Chicago	United States
11	Los Angeles	United States
10	New York	United States

Procedury

update_trip_price

Procedura pozwalająca dla wybranej wycieczki (trip_id) zmienić jej cenę za osobę (new_price).

```
create procedure update_trip_price
    @trip_id int,
    @new_price money
as
begin
    update Trips
    set Price_per_person = @new_price
    where Trip_id = @trip_id;
end;
```

Test działania:

```
select Trip_id, Price_per_person from trips
where Trip_id = 1;
```

Trip_id	Price_per_person
1	500,00

```
exec update_trip_price 1, 600;
select Trip_id, Price_per_person from trips
where Trip_id = 1;
```

Trip_id	Price_per_person
1	600,00

update_trip_limit

Procedura pozwalająca dla wybranej wycieczki (trip_id) zmienić limit jej uczestników (new_limit)

```
create procedure update_trip_limit
    @trip_id int,
    @new_limit int
as
begin
    update Trips
    set Limit = @new_limit
    where Trip_id = @trip_id;
end;
```

Test działania:

```
select trip_id, Limit from trips
where trip_id = 1;
```

trip_id	Limit
1	20

```
exec update_trip_limit 1, 15;
select trip_id, Limit from trips
where trip_id = 1;
```

trip_id	Limit
1	15

update_attraction_price

Procedura pozwalająca dla wybranej wycieczki (attraction_id) zmienić cenę za osobę (new_price).

```
create procedure update_attraction_price
    @attraction_id int,
    @new_price money
as
begin
    update Attractions
    set Price_per_person = @new_price
    where Attraction_id = @attraction_id;
end;
```

update_attraction_limit

Procedura pozwalająca dla wybranej wycieczki (attraction_id) zmienić limit jej uczestników(new_limit).

```
create procedure update_attraction_limit
    @attraction_id int,
    @new_limit int
as
begin
    update Attractions
    set Limit = @new_limit
    where Attraction_id = @attraction_id;
end;
```

update_trip_date

Procedura pozwalająca na zaaktualizowanie daty odbycia się wycieczki (pole Date).

```
create or alter procedure update_trip_date
(
    @Trip_id int,
    @Date date
)
as
begin
    update Trips
    set Date = @Date
    where Trip_id = @Trip_id;
end;
```

Test działania

```
exec update_trip_date
    @Trip_id = 1,
    @Date = '2024-06-17'
```

Wygląd rekordu Trip_id = 1, po wprowadzonych zmianach.

Trip_id	Name	City_id	Date	Price_per_person	Limit	Start_reservation
1	Discover Paris	4	2024-06-17	600.0000	20	2024-05-15 10:00:00.000

update_trip_start_reservation

Procedura umożliwia zaaktualizowanie daty rozpoczęcia rezerwacji.

```
create or alter procedure update_trip_start_reservation
(
    @Trip_id int,
    @Start_reservation datetime
)
as
begin
    update Trips
    set Start_reservation = @Start_reservation
    where Trip_id = @Trip_id;
end;
```

Test działania

```
exec update_trip_start_reservation
    @Trip_id = 1,
    @Start_reservation = '2024-05-15 10:00:00.000'
```

Wygląd rekordu Trip_id = 1, po wprowadzonych zmianach.

Trip_id	Name	City_id	Date	Price_per_person	Limit	Start_reservation
1	Discover Paris	4	2024-06-17	600.0000	20	2024-05-15 08:00:00.000

add_new_client

Procedura umożliwiająca dodanie nowego klienta do bazy danych.

```
create or alter procedure add_new_client
    @Email varchar(100),
    @Phone varchar(9),
```

```
    @Client_type CHAR(1), -- 'C' dla osoby fizycznej, 'P' dla firmy
    @Firstname varchar(40) = null,
    @Lastname varchar(40) = null,
    @Company_name varchar(50) = null,
    @NIP int = null
as
begin
    set nocount on;

    -- Sprawdzanie warunków przed rozpoczęciem transakcji
    if @Client_type = 'C' and (@Firstname is null or @Lastname is null)
    begin
        raiserror('Firstname and Lastname are required for Customers.', 16, 1);
        return;
    end

    if @Client_type = 'P' and (@Company_name is null or @NIP is null)
    begin
        raiserror('CompanyName and NIP are required for Companies.', 16, 1);
        return;
    end

    if @Client_type not in ('C', 'P')
    begin
        raiserror('Invalid Client_type. Use ''C'' for Customers or ''P'' for Companies.', 16, 1);
        return;
    end

    begin transaction;

    begin try
        declare @Client_id int;

        -- Dodawanie nowego klienta do tabeli Clients
        insert into Clients (Email, Phone)
        values (@Email, @Phone);

        -- Pobieranie ID nowo dodanego klienta
        set @Client_id = SCOPE_IDENTITY();

        -- Dodawanie klienta do tabeli Customers lub Companies na podstawie typu klienta
        if @Client_type = 'C'
        begin
            insert into Customers (Client_id, Firstname, Lastname)
            values (@Client_id, @Firstname, @Lastname);
        end
        else if @Client_type = 'P'
        begin
            insert into Companies (Client_id, Company_name, NIP)
            values (@Client_id, @Company_name, @NIP);
        end

        -- Jeśli wszystko się powiodło, zatwierdź transakcję
        commit transaction;
    end try
    begin catch
        -- Jeśli wystąpił błąd, wycofaj transakcję
        if @@TRANCOUNT > 0
            rollback transaction;

        -- Rzuć błąd ponownie, aby informować o problemie
        throw;
    end catch
end;
```

Testy sprawdzające działanie procedury.

1. Dodanie nowego klienta fizycznego. W tym celu należy podać:
    - Email
    - Phone - numer telefonu
    - Client_type - informacja o tym czy wprowadzany klient jest osobą fizyczną (C) lub firmą (P)
    - Firstname - imię klienta
    - Lastname - nazwisko klienta

```
exec add_new_client
    @Email = 'john.doe@example.com',
    @Phone = '123456789',
    @Client_type = 'C',
    @Firstname = 'John',
    @Lastname = 'Doe';
```

Wynik wykonania powyższej procedury:

- tabela Clients

```
select * from Clients
where Email = 'john.doe@example.com'
```

Client_id	Email	Phone
21	john.doe@example.com	123456789

- tabela Clients + Customers

```
select cu.Client_id, FirstName, Lastname, Email, Phone from Clients cl
join Customers cu on cu.Client_id=cl.Client_id
where Email = 'john.doe@example.com';
```

Client_id	Firstname	Lastname	Email	Phone
21	John	Doe	john.doe@example.com	123456789

2. Dodanie nowej firmy. W tym celu należy podać:
   - Email
   - Phone - numer telefonu
   - Client_type - informacja o tym czy wprowadzany klient jest osobą fizyczną (C) lub firmą (P)
   - Companyname - nazwa firmy
   - NIP - nr NIP

```
exec add_new_client
    @Email = 'contact@microsoft.com',
    @Phone = '987654321',
    @Client_type = 'P',
    @Company_name = 'Microsoft',
    @NIP = 1234567890;
```

Wynik wykonania powyższej procedury:

- tabela Clients

```
select * from Clients
where Email = 'contact@microsoft.com'
```

Client_id	Email	Phone
22	contact@microsoft.com	987654321

- tabela Clients + Customers

```
select co.Client_id, Company_name, NIP, Email, Phone from Clients cl
join Companies co on co.Client_id=cl.Client_id
where Email = 'contact@microsoft.com';
```

Client_id	Company_name	NIP	Email	Phone
22	Microsoft	123456789	contact@microsoft.com	987654321

add_new_trip

Procedura ta umożliwia dodawanie nowej wycieczki. Uwzględniono rówież sytuację, gdy dodawana jest wycieczka do nowego kraju lub miasta.

```
create or alter procedure add_new_trip
(
    @Name varchar(40),
    @City varchar(50),
    @Country varchar(50),
    @Date date,
    @Price_per_person money,
    @Limit int,
    @Start_reservation datetime
)
as
begin
    declare @City_id int;
    declare @CountryID int;

    -- Sprawdzenie czy kraj już istnieje, jeśli nie, dodaj nowy kraj
    if not EXISTS (select * from Countries where Country = @Country)
    begin
        insert into Countries (Country)
        values (@Country);
```

```
        set @CountryID = SCOPE_IDENTITY();
    end
    else
    begin
        select @CountryID = Country_id
        from Countries
        where Country = @Country;
    end

    -- Sprawdzenie czy miasto już istnieje, jeśli nie, dodaj nowe miasto
    if not EXISTS (select * from Cities where City = @City and Country_id = @CountryID)
    begin
        insert into Cities (City, Country_id)
        values (@City, @CountryID);

        set @City_id = SCOPE_IDENTITY();
    end
    else
    begin
        select @City_id = City_id
        from Cities
        where City = @City and Country_id = @CountryID;
    end

    -- Dodanie nowej wycieczki
    insert into Trips (Name, City_id, Date, Price_per_person, Limit, Start_reservation)
    values (@Name, @City_id, @Date, @Price_per_person, @Limit, @Start_reservation);
end;
```

Testy działania:

1. Dodanie wycieczki do istniejącego w bazie danych miasta (Paryż) oraz państwa (Francja).

```
exec add_new_trip
    @Name = 'Sightseeing in paris',
    @City = 'Paris',
    @Country = 'France',
    @Date = '2024-07-22',
    @Price_per_person = '700.0000',
    @Limit = 15,
    @Start_reservation = '2024-06-22 2:00:00.000'
```

   Po dodaniu nowej wycieczki, widzimy zmiany w tabeli Trips.

Trip_id	Name	City_id	Date	Price_per_person	Limit	Start_reservation
9	Sightseeing in paris	4	2024-07-22	700.0000	15	2024-06-22 2:00:00.000

2. Dodanie wycieczki do nowego miasta (Neapol), ale istniejącego już w bazie danych państwa (Włochy).

```
exec add_new_trip
@Name = 'Travel to Naples',
@City = 'Naples',
@Country = 'Italy',
@Date = '2024-08-20',
@Price_per_person = '450.0000',
@Limit = 7,
@Start_reservation = '2024-06-11 12:00:00.000'
```

   Po dodaniu nowej wycieczki, widzimy zmiany w tabeli Trips.

Trip_id	Name	City_id	Date	Price_per_person	Limit	Start_reservation
10	Travel to Naples	31	2024-08-20	450.0000	7	2024-06-11 12:00:00.000

3. Dodanie wycieczki do nowego miasta (Kraków) oraz nowego państwa (Polska).

```
exec add_new_trip
@Name = 'Walk around Krakow',
@City = 'Krakow',
@Country = 'Poland',
@Date = '2024-10-15',
@Price_per_person = '200.0000',
@Limit = 20,
@Start_reservation = '2024-08-13 05:00:00.000'
```

   Po dodaniu nowej wycieczki, widzimy zmiany w tabeli Trips.

Trip_id	Name	City_id	Date	Price_per_person	Limit	Start_reservation
11	Walk around Krakow	32	2024-10-15	200.0000	20	2024-08-13 05:00:00.000

add_new_attraction

Procedura jest odpowiedzialna za dodanie nowej atrakcji do istniejącej wycieczki.

```
create or alter procedure add_new_attraction
(
    @Trip_id int,
    @Name varchar(50),
    @Price_per_person money,
    @Limit int
)
as
begin
    insert into Attractions (Trip_id, Name, Price_per_person, Limit)
    values (@Trip_id, @Name, @Price_per_person, @Limit);
end;
```

Test działania

```
exec add_new_attraction
    @Trip_id = 10,
    @Name = 'Climbing Vesuvius',
    @Price_per_person = 60.0000,
    @Limit = 3
```

Po dodaniu nowej atrakcji dla wycieczki (Trip_id = 10), widzimy zmiany w tabeli Attractions.

Attraction_id	Trip_id	Name	Price_per_person	Limit
25	11	Climbing Vesuvius	60.0000	3

update_client

```
create or alter procedure update_client
(
    @Client_id int,
    @New_value varchar(100),
    @Update_type char(1) -- 'E' dla Email, 'P' dla Phone
)
as
begin
    if @Update_type = 'E'
    begin
        update Clients
        set Email = @New_value
        where Client_id = @Client_id;
    end
    else if @Update_type = 'P'
    begin
        update Clients
        set Phone = @New_value
        where Client_id = @Client_id;
    end
    else
    begin
        raiserror('Invalid Update_type. Use ''E'' for Email or ''P'' for Phone.', 16, 1);
    end
end;
```

Testy działania:

- zmiana maila klienta

  ```
  exec update_client
      @Client_id = 11,
      @New_value = 'kowalski@onet.pl',
      @Update_type = 'E'
  ```

Client_id	Email	Phone
11	kowalski@onet.pl	741852963

- zmiana nr telefonu firmy

  ```
  exec update_client
  @Client_id = 1,
  @New_value = '794551123',
  @Update_type = 'P'
  ```

Client_id	Email	Phone
1	techinnovators@example.com	794551123

add_new_reservation

Procedura dodaje nową rezerwację dla podanego klienta

```
create or alter procedure add_new_reservation
(
    @Trip_id int,
    @Client_id int,
    @Count_of_trippers int,
    @Cancelled bit = 0
)
as
begin
    declare @Price_per_person money;
    select @Price_per_person = Price_per_person from Trips where Trip_id = @Trip_id;

    insert into Reservations (Trip_id, Client_id, Price_per_person, Count_of_trippers, Cancelled)
    values (@Trip_id, @Client_id, @Price_per_person, @Count_of_trippers, @Cancelled);
end;
```

Test działania:

```
exec add_new_reservation
    @Trip_id = 8,
    @Client_id = 1,
    @Count_of_trippers = 1
```

W tabeli Reservations możemy dostrzec nową rezerwację dla klienta o Client_id = 1.

Rervation_id	Trip_id	Client_id	Price_per_person	Count_of_trippers	Cancelled
12	8	1	1200.0000	1	false

cancel_reservation

Procedura umożliwia anulowanie danej rezerwacji

```
create or alter procedure cancel_reservation
(
    @Reservation_id int
)
as
begin
    begin transaction ;

    begin try
        -- Oznacz rezerwację jako anulowaną
        update Reservations
        set Cancelled = 1
        where Reservation_id = @Reservation_id;

        commit transaction ;
    end try
    begin catch
        if @@TRANCOUNT > 0
            rollback transaction ;
        throw;
    end catch;
end;
```

Test działania

```
exec cancel_reservation
    @Reservation_id = 1
```

Rezerwacja (Reservation_id = 1) dla wycieczki (Trip_id = 1) została anulowana.

Reservation_id	Trip_id	Client_id	Price_per_person	Count_of_trippers	Cancelled
1	1	15	500.0000	1	true

add_new_attraction_reservation

Procedura umożliwia dodanie nowej rezerwacji atrakcji.

```
create or alter procedure add_new_attraction_reservation
(
    @Reservation_id int,
    @Attraction_id int,
    @Count_of_trippers int
)
as
begin
    declare @Price_per_person money;
    select @Price_per_person = Price_per_person from Attractions where Attraction_id = @Attraction_id;

    insert into Attraction_Reservations (Price_per_person, Count_of_trippers, Attraction_id, Reservation_id)
    values (@Price_per_person, @Count_of_trippers, @Attraction_id, @Reservation_id);
end;
```

Test działania Dodanie nowej rezerwacji atrakcji nr 7, dla jednego (Count_of_trippers = 1) uczestnika.

```
exec add_new_attraction_reservation
    @Reservation_id = 3,
    @Attraction_id = 7,
    @Count_of_trippers = 1
```

Zmiany wprowadzone w tabeli Attraction_Reservations wyglądają następująco.

AR_id	Price_per_person	Count_of_trippers	Reservation_id	Attraction_id
1	40.0000	1	1	2
2	80.0000	1	3	7
3	120.0000	4	5	17
4	80.0000	1	3	7

add_new_payment

Procedura odpowiedzialna jest za dodawanie nowej płatności

```
create or alter procedure add_new_payment
(
    @Paid money,
    @Reservation_id int,
    @Method char(1)
)
as
begin
    declare @Timestamp datetime;

    select @Timestamp = CAST(SYSDATETIME() as datetime)

    insert into Payments (Paid, Timestamp, Reservation_id, Method)
    values (@Paid, @Timestamp, @Reservation_id, @Method);
end;
```

Test działania:

```
exec add_new_payment
    @Paid = 2000,
    @Reservation_id = 6,
    @Method = 'C'
```

Dodano nową płatność, co dostrzec możemy w tabeli Payments.

Payment_id	Paid	Timestamp	Reservation_id	Method
5	2000.0000	2024-05-29 22:18:16.617	6	C

update_person_data

Procedura odpowiedzialna jest za zmianę danych w tabeli People, dla poszczególnej osoby.

```
create or alter procedure update_person_data
(
    @Person_id int,
    @Update_type char(1), -- 'E' dla Email, 'P' dla Phone
    @Email varchar(100) = null,
    @Phone varchar(9) = null
)
as
begin
    if @Update_type = 'E'
```

```
    begin
        update People
        set Email = @Email
        where Person_id = @Person_id;
    end
    else if @Update_type = 'P'
    begin
        update People
        set Phone = @Phone
        where Person_id = @Person_id;
    end
end;
```

Test działania

```
exec update_person_data
    @Person_id = 1,
    @Update_type = 'P',
    @Phone = '234567890'
```

Dokonano zmiany nr telefonu dla osoby (Person_id = 1).

Person_id	Firstname	Lastname	Email	Phone
1	Adam	Kowalski	adam.kowalski@example.com	234567890

Funkcje

PaymentDetails

Funkcja wypisuje szczegóły płatności w podanym zakresie dat.

```
create function PaymentDetails
(
    @start_date date,
    @end_date date
)
returns table
as
return
(select
    pa.payment_id,
    pa.paid,
    pa.timestamp,
    pa.method,
    r.reservation_id,
    c.client_id,
    case
        when cust.customer_id is not null then cust.firstname + ' ' + cust.lastname
        when comp.company_id is not null then comp.company_name
    end as clientname,
    t.trip_id,
    t.name as tripname
from payments pa
join reservations r on r.reservation_id = pa.reservation_id
join clients c on c.client_id = r.client_id
left join customers cust on cust.client_id = c.client_id
left join companies comp on comp.client_id = c.client_id
left join trips t on t.trip_id = r.trip_id
where pa.timestamp between @start_date and @end_date);
```

Test działania

```
select * from Payment_Details('2024-05-01', '2024-05-20');
```

Płatności dokonane między 1 maja 2024 a 20 maja 2024 rokiem.

payment_id	paid	timestamp	method	reservation_id	client_id	clientname	trip_id	tripname
1	540.0000	2024-05-19 18:31:34.000	M	1	15	Andrzej Kowalczyk	1	Discover Paris
2	3120.0000	2024-05-19 18:31:34.000	M	2	5	Future Builders	3	Adventures in Tokyo
3	2600.0000	2024-05-19 18:31:34.000	C	4	18	Joanna Zielinska	4	Safari in Guadalajara
4	7200.0000	2024-05-19 18:31:34.000	P	5	20	Magdalena Wozniak	6	Hike in Machu Picchu

SearchAttractionsInCity

Funkcja umożliwia wyszukiwanie atrakcji na podstawie podania City_id

```
```

```
create or alter function SearchAttractionsInCity (@CityID INT)
returns table
as
return
(select
    a.Attraction_id,
    a.Name,
    c.City,
    a.Price_per_person,
    COALESCE(a.Limit - SUM(at.Count_of_trippers), a.Limit) Limit
from Attractions a
join Trips t on a.Trip_id = t.Trip_id
join Cities c on c.City_id=t.City_id
left join Attraction_Reservations at on at.Attraction_id=a.Attraction_id
where t.City_id = @CityID
group by a.Attraction_id, a.Name, c.City, a.Price_per_person, a.Limit);
```

Test działania funkcji

- atrakcje dla wycieczki do Paryża, stolicy Francji

```
select * from SearchAttractionsInCity(4)
```

Attraction_id	Name	City	Price_per_person	Limit
1	Eiffel Tower Tour	Paris	50.0000	15
2	Louvre Museum Visit	Paris	40.0000	9
3	Seine River Cruise	Paris	60.0000	5

- atrakcje dla wycieczki do Lima, stolicy Peru

```
select * from SearchAttractionsInCity(29)
```

Attraction_id	Name	City	Price_per_person	Limit
16	Machu Picchu Guided Tour	Lima	80.0000	20
17	Inca Trail Trek	Lima	120.0000	11
18	Huayna Picchu Summit	Lima	100.0000	10

TripsWithRange

Funkcja wypisuje wycieczki w podanym zakresie dat.

```
create function TripsWithRange
(
    @start_date date,
    @end_date date
)
returns table
as
return
(select
    trip_id,
    name,
    city_id,
    date,
    price_per_person,
    limit,
    start_reservation
from trips
where date between @start_date and @end_date);
```

Test działania

```
select * from TripsWithRange('2024-06-01', '2024-06-30');
```

Wynikiem są wycieczki organizowane między 2024-06-01 a 2024-06-30.

trip_id	name	city_id	date	price_per_person	limit	start_reservation
1	Discover Paris	4	2024-06-17	600.0000	20	2024-05-15 10:00:00.000
9	Walk around Krakow	31	2024-06-15	600.0000	5	2024-05-15 08:00:00.000

GetClientsForTrip

Funkcja wypisuje klientów, którzy zarezerwowali wybraną wycieczkę.

```
create or alter function GetClientsForTrip
(
    @trip_id int
)
returns table
as
return
(select
    c.client_id,
    case
        when cust.customer_id is not null then cust.firstname + ' ' + cust.lastname
        when comp.company_id is not null then comp.company_name
    end as clientname,
    t.trip_id,
    t.name as tripname
from clients c
join reservations r on c.Client_id = r.Client_id
left join customers cust on cust.client_id = c.client_id
left join companies comp on comp.client_id = c.client_id
left join trips t on t.trip_id = r.trip_id
where r.trip_id = @trip_id);
```

Test działania

```
select * from GetClientsForTrip(1)
```

Jak się okazuje wycieczkę (Trip_id = 1) zarezerwował klient (Client_id = 15)

client_id	clientname	trip_id	tripname
15	Andrzej Kowalczyk	1	Discover Paris

GetParticipantsForTrip

Funkcja wypisuje uczestników wybranej wycieczki.

```
create or alter function GetParticipantsForTrip
(
    @trip_id int
)
returns table
as
return
(select
    t.Tripper_id,
    p.Firstname,
    p.Lastname
from Trippers t
join People p on t.Person_id = p.Person_id
where t.Reservation_id
in (select Reservation_id from Reservations where Trip_id = @trip_id));
```

Test działania

```
select * from GetParticipantsForTrip(3);
```

Dla wycieczki (Trip_id = 3) wynikiem jest poniższa lista uczestników.

Tripper_id	Firstname	Lastname
2	Jan	Nowak
3	Marta	Kowalska
4	Tadeusz	Wisniewski
5	Katarzyna	Wójcik

GetParticipantsForAttraction

Funkcja wypisuje uczestników wybranej atrakcji

```
CREATE FUNCTION GetParticipantsForAttraction(@attraction_id INT)
    RETURNS TABLE
        AS
        RETURN
        (
            SELECT P.Person_id ,P.Firstname, P.Lastname
```

```
            FROM People P
            JOIN Trippers T ON P.Person_id = T.Person_id
            JOIN dbo.Attractions_Trippers A on T.Tripper_id = A.Tripper_id
            JOIN dbo.Attraction_Reservations AR on A.AR_id = AR.AR_id
            JOIN dbo.Attractions AT on AR.Attraction_id = AT.Attraction_id
            WHERE AT.Attraction_id = @attraction_id
        );
```

Test działania

```
select * from GetParticipantsForAttraction(7);
```

Person_id	Firstname	Lastname
21	Jan	Nowak
22	Marta	Kowalska
23	Tadeusz	Wisniewski
24	Katarzyna	Wójcik

GetClientDetails

Funkcja wypisuje szczegółowe informacje o kliencie bazy danych, poprzez podanie jako argument Client_id

```
create or alter function GetClientDetails(@ClientID INT)
returns table
as
return
(select
    c.Client_id,
    case
        when cu.customer_id is not null then cu.firstname + ' ' + cu.lastname
        when co.company_id is not null then co.company_name
    end as Client_name,
    c.Email,
    c.Phone
from Clients c
left join Customers cu ON c.Client_id = cu.Client_id
left join Companies co ON c.Client_id = co.Client_id
where c.Client_id = @ClientID);
```

Działanie funkcji, przedstawiają dwa poniższe przykłady:

- dla firmy

  ```
  select * from GetClientDetails(1)
  ```

Client_id	Client_name	Email	Phone
1	Tech Innovators	techinnovators@example.com	123456789

- dla osoby fizycznej

  ```
  select * from GetClientDetails(11)
  ```

Client_id	Client_name	Email	Phone
11	Jan Kowalski	kowalski@example.com	741852963

GetReservationsForClient

Funkcja wypisuje rezerwacje dla podanego klienta

```
create or alter function GetReservationsForClient(@Client_id int)
returns table
as
return
(select
    r.Reservation_id,
    r.Trip_id,
    t.Name AS Trip_name,
    r.Price_per_person,
    r.Count_of_trippers,
    r.Cancelled
from Reservations r
join Trips t ON r.Trip_id = t.Trip_id
where r.Client_id = @Client_id);
```

Test działania

```
select * from GetReservationsForClient(5)
```

Wynkiem są rezerwacje dla kienta (Client_id = 5).

Client_id	Reservation_id	Trip_id	Trip_name	Price_per_person	Count_of_trippers	Cancelled
5	2	3	Adventures in Tokyo	700.0000	4	false

GetAvailableAttractionsForTrip

Funkcja umożliwia wyszukanie dostępnych atrakcji dla podanej wycieczki.

```
create or alter function GetAvailableAttractionsForTrip(@Trip_id int)
returns table
as
return
(select
    a.Attraction_id,
    a.Name,
    a.Price_per_person,
    a.Limit - COALESCE(SUM(ar.Count_of_trippers), 0) Available_spots
from Attractions a
left join Attraction_Reservations ar on a.Attraction_id = ar.Attraction_id
where a.Trip_id = @Trip_id
group by a.Attraction_id, a.Name, a.Price_per_person, a.Limit
);
```

Test działania

```
select * from GetAvailableAttractionsForTrip(1)
```

Attraction_id	Name	Price_per_person	Available_spots
1	Eiffel Tower Tour	50.0000	15
2	Louvre Museum Visit	40.0000	9
3	Seine River Cruise	60.0000	5

Powyższa funkcja uwzględniła również zarezerwowane atrakcje

```
select AR_id, Count_of_trippers from Attraction_Reservations
where Attraction_id = 2
```

AR_id	Count_of_trippers
1	1

```
select Attraction_id, Limit from Attractions
where Attraction_id = 2
```

Attraction_id	Limit
2	10

GetCancelledTripsForClient

Funkcja wypisuje wszystkie wycieczki klienta, które zostały anulowane.

```
create or alter function GetCancelledTripsForClient(@Client_id int)
returns table
as
return
(select
    r.Reservation_id,
    r.Trip_id,
    t.Name TripName,
    r.Price_per_person,
    r.Count_of_trippers,
    r.Cancelled
from Reservations r
join Trips t on r.Trip_id = t.Trip_id
where r.Client_id = @Client_id and r.Cancelled = 1);
```

Test działania

```
select * from GetCancelledTripsForClient(1)
```

Wynkiem są anulowane wycieczki dla kienta (Client_id = 1).

Reservation_id	Trip_id	TripName	Price_per_person	Count_of_trippers	Cancelled
13	8	Relaxing in Athens	1200.0000	1	true

Triggery

CheckTrippersCount

Trigger ma za zadanie sprawdzić, czy ilość rekordów w tabeli trippers powiązanych z daną rezerwacją jest mniejsza lub równa ilości zadeklarowanej przez osobę składającą rezerwację

```
CREATE TRIGGER CheckTrippersCount
    ON Trippers
    AFTER INSERT, UPDATE
    AS
BEGIN
    IF EXISTS (
        SELECT 1
        FROM inserted i
                 JOIN (
            SELECT Reservation_id, COUNT(*) as TrippersCount
            FROM Trippers
            WHERE Reservation_id IN (SELECT Reservation_id FROM inserted)
            GROUP BY Reservation_id
        ) T ON i.Reservation_id = T.Reservation_id
                 JOIN Reservations R ON i.Reservation_id = R.Reservation_id
        WHERE T.TrippersCount > R.Count_of_trippers
    )
        BEGIN
            RAISERROR ('Liczba rekordów w tabeli Trippers nie może przekraczać wartości Count_of_trippers w tabeli Reservations.',
16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END;
END;
```

CheckAttractionTrippersCount

Tożsame do poprzedniego triggera, ale sprawdzającą czy ilość zadeklarowanych uczestników dla danej atrakcji jest mniejsza lub równa ilości podanej w rezerwacji.

```
CREATE OR ALTER TRIGGER CheckAttractionTrippersCount
        ON Attractions_Trippers
        AFTER INSERT, UPDATE
        AS
    BEGIN
        IF EXISTS (
            SELECT 1
            FROM inserted i
                     JOIN (
                SELECT AR_id, COUNT(*) as AttractionTrippersCount
                FROM Attractions_Trippers
                WHERE AR_id IN (SELECT AR_id FROM inserted)
                GROUP BY AR_id
            ) T ON i.AR_id = T.AR_id
                     JOIN Attraction_Reservations AR ON i.AR_id = AR.AR_id
            WHERE T.AttractionTrippersCount > AR.Count_of_trippers
        )
            BEGIN
                RAISERROR ('Liczba rekordów w tabeli Attraction_Trippers nie może przekraczać wartości Count_of_trippers w tabeli
Attraction_Reservations.', 16, 1);
                ROLLBACK TRANSACTION;
                RETURN;
            END;
    END;
```

CheckTrippersLimit

Trigger ma za zadanie sprawdzić czy łączna ilość uczestników zadeklarowanych we wszystkich rezerwacjach dotyczących wycieczki jest mniejsza lub równa od limitu uczestników danej wycieczki.

```
CREATE OR ALTER TRIGGER CheckTrippersLimit
    ON Reservations
    AFTER INSERT, UPDATE
    AS
BEGIN
```

```
    IF EXISTS (
        SELECT 1
        FROM inserted i
                 JOIN (
            SELECT Trip_id, SUM(Count_of_trippers) as TotalTrippers
            FROM Reservations
            WHERE Trip_id IN (SELECT Trip_id FROM inserted)
            GROUP BY Trip_id
        ) R ON i.Trip_id = R.Trip_id
                 JOIN Trips T ON i.Trip_id = T.Trip_id
        WHERE R.TotalTrippers > T.limit
    )
        BEGIN
            RAISERROR ('Suma Count_of_trippers w tabeli Reservations nie może przekraczać wartości limit w powiązanym rekordzie z
tabeli Trips.', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END;
END;
```

CheckAttractionTrippersLimit

Trigger ma za zadanie sprawdzić czy łączna ilość uczestników zadeklarowanych we wszystkich rezerwacjach atrakcji dotyczących wycieczki jest mniejsza lub równa od limitu uczestników danej atrakcji.

```
CREATE OR ALTER TRIGGER CheckAttractionTrippersLimit
    ON Attraction_Reservations
    AFTER INSERT, UPDATE
    AS
BEGIN
    IF EXISTS (
        SELECT 1
        FROM inserted i
                 JOIN (
            SELECT Attraction_id, SUM(Count_of_trippers) as TotalTrippers
            FROM Attraction_Reservations
            WHERE Attraction_id IN (SELECT Attraction_id FROM inserted)
            GROUP BY Attraction_id
        ) R ON i.Attraction_id = R.Attraction_id
                 JOIN Attractions A ON i.Attraction_id = A.Attraction_id
        WHERE R.TotalTrippers > A.limit
    )
        BEGIN
            RAISERROR ('Suma Count_of_trippers w tabeli Attraction_Reservations nie może przekraczać wartości limit w powiązanym
rekordzie z tabeli Attractions.', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END;
END;
```

4. Inne

4.1 Informacje dotyczące wygenerowanych danych

Dane do bazy danych wprowadzamy przez przygotowany skrypt sql. Podczas wprowadzania pierwszych danych nie wykorzystujemy opcji automatycznej inkrementacji kluczy. Dane - mające symulować te prawdziwe - pochodzą z generatorów fałszywych tożsamośći, zostały wygenerowane przez modele językowe lub zostały wymyślone przez nas. Składają się one m.in. z: 8 różnych wycieczek, 24 dodatkowych atrakcji, 20 klientów(10 firm + 10 osób fizycznych), 35 uczestników wycieczek, 10 krajów oraz 30 miast.