Bazy danych

Autorzy: Aleksandra Rząca, Arina Pashkevich, Kacper Góra, Paweł Rydz

1. Opis systemu

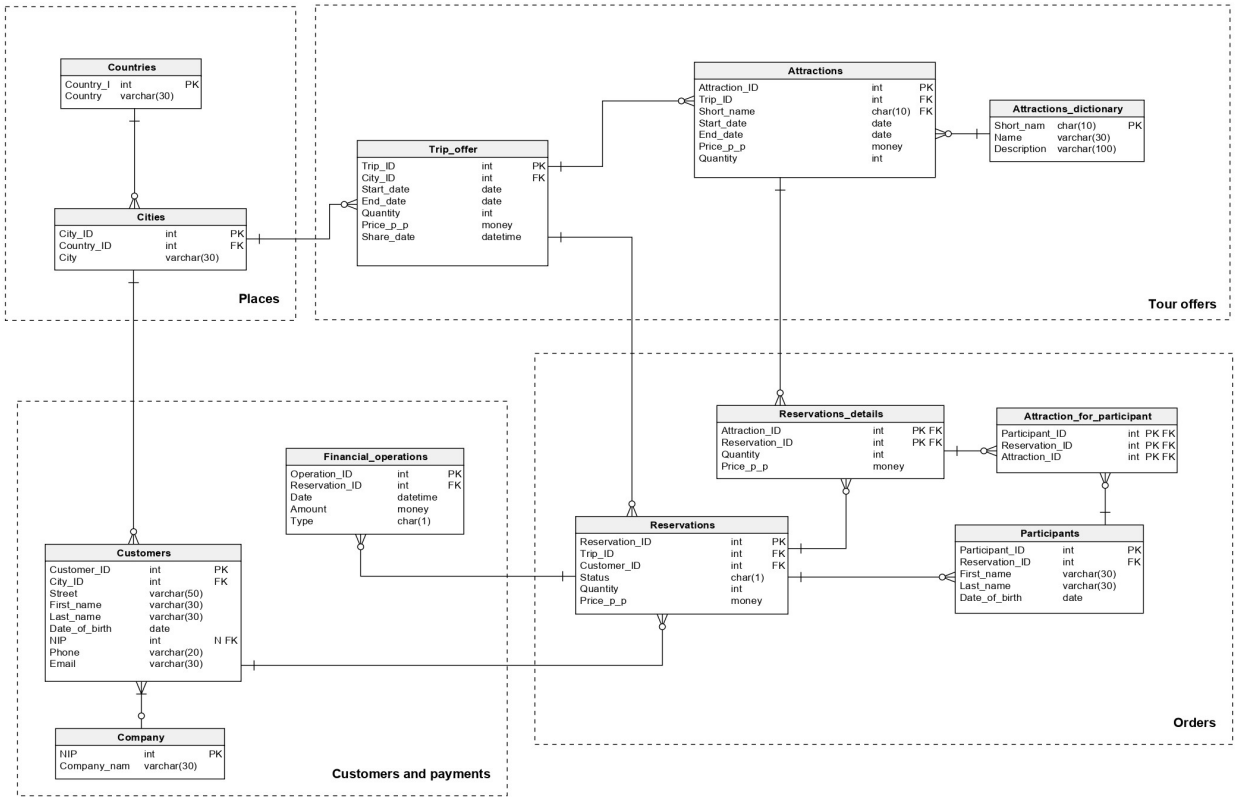
Baza danych biura podróży ma na celu efektywne zarządzanie ofertami, rezerwacjami i płatnościami w ramach działalności biura. Reprezentuje system informatyczny, który umożliwia administratorom i pracownikom tworzenie, edytowanie i monitorowanie ofert wycieczek, dodając atrakcje i zarządzając rezerwacjami. Klienci mogą przeglądać oferty, składać rezerwacje oraz zarządzać nimi. System obsługuje także procesy płatności, umożliwiając śledzenie wpłat i wypłat oraz generowanie raportów finansowych dla administratorów. Dzięki temu biuro podróży może efektywnie zarządzać swoją działalnością, zapewniając klientom łatwy dostęp do informacji i usług.

Wymagania i funkcje

- Zakładanie oferty wycieczki (Administrator, Pracownik)
- Dodawanie atrakcji do danej wycieczki (Administrator, Pracownik)
- Wyświetlenie oferty wycieczek wraz z atrakcjami (wszyscy)
- Składanie rezerwacji (Klient, Pracownik)
- Edytowanie rezerwacji (Klient, Pracownik)
- Wyświetlanie szczegółów rezerwacji (Klient, Pracownik)
- Zatwierdzenie zamówienia (Agent_systemowy, Administrator)
- Anulowanie zamówienia (Klient, Agent_systemowy, Pracownik, Administrator)
- Wyświetlenie zamówienia (Klient, Pracownik, Administrator)
- Dodawanie uczestników do danej rezerwacji (Klient, Pracownik, Administrator)
- Edytowanie uczestników danej wycieczki (Klient, Pracownik, Administrator)
- Wyświetlenie uczestników danej wycieczki (Klient, Pracownik, Administrator)
- Rejestrowanie wpłat dotyczących odpowiedniej rezerwacji (Agent_systemowy)
- Zwracanie nadpłaty (Pracownik, Administrator)
- Wyświetlenie stanu konta dla każdego użytkownika (Klient, Administrator)
- Wyświetlenie raportu wpłat i wypłat dla danego okresu (Administrator)

2. Baza danych

Schemat bazy danych



Opis poszczególnych tabel

Nazwa tabeli: Trip_offer

- Opis: Tabela zawiera główne oferty biura podróży, w tym informacje o miejscu wycieczki oraz szczegóły dotyczące dostępności i ceny.

Nazwa atrybutu	Typ	Opis/Uwagi
Trip_ID	PK int	Identyfikator wycieczki.
City_ID	FK int	Identyfikator miasta, do którego jest wycieczka.
Start_date	date	Data rozpoczęcia wycieczki.
End_date	date	Data zakończenia wycieczki.
Quantity	int	Liczba oferowanych miejsc na wycieczkę.
Price_p_p	money	Cena wycieczki za jedną osobę.
Share_date	datetime	Data udostępnienia oferty wycieczki dla użytkowników (po tej dacie oferta jest dostępna do wyświetlenia).

- kod DDL

```
-- Table: Trip_offer
CREATE TABLE Trip_offer (
    Trip_ID int IDENTITY(1,1)  NOT NULL,
    City_ID int  NOT NULL,
    Start_date date  NOT NULL,
    End_date date  NOT NULL,
    Quantity int  NOT NULL,
    Price_p_p money  NOT NULL,
    Share_date datetime  NOT NULL,
    CONSTRAINT Trip_ID PRIMARY KEY  (Trip_ID),
    CONSTRAINT chk_start_date_for_trip CHECK(Start_date > DATEADD(day, 7, GETDATE())),
    CONSTRAINT chk_end_date_for_trip CHECK(End_date >= Start_date),
    CONSTRAINT chk_quantity_for_trip CHECK(Quantity > 0),
    CONSTRAINT chk_price_for_trip CHECK(Price_p_p > 0),
    CONSTRAINT chk_share_date_for_trip CHECK(Share_date <= DATEADD(day, -7, Start_date))
);
-- Reference: Trip_offer_Cities (table: Trip_offer)
ALTER TABLE Trip_offer ADD CONSTRAINT Trip_offer_Cities
    FOREIGN KEY (City_ID)
    REFERENCES Cities (City_ID);
```

Nazwa tabeli: Countries

- Opis: Tabela słownikowa zawierająca informacje o krajach.

Nazwa atrybutu	Typ	Opis/Uwagi
Country_ID	PK int	Identyfikator kraju.
Country	varchar(30)	Nazwa kraju.

- kod DDL

```
-- Table: Countries
CREATE TABLE Countries (
    Country_ID int IDENTITY(1,1) NOT NULL,
    Country varchar(30)  NOT NULL,
    CONSTRAINT Countries_pk PRIMARY KEY  (Country_ID)
);
```

Nazwa tabeli: Cities

- Opis: Tabela słownikowa zawierająca informacje o miastach.

Nazwa atrybutu	Typ	Opis/Uwagi
City_ID	PK int	Identyfikator miasta.
Country_ID	FK int	Identyfikator kraju, w którym znajduje się miasto.
City	PK varchar(30)	Nazwa miasta.

- kod DDL

```
-- Table: Cities
CREATE TABLE Cities (
    City_ID int IDENTITY(1,1) NOT NULL,
    Country_ID int  NOT NULL,
    City varchar(30)  NOT NULL,
    CONSTRAINT Cities_pk PRIMARY KEY  (City_ID)
);
-- Reference: Cities_Countries (table: Cities)
ALTER TABLE Cities ADD CONSTRAINT Cities_Countries
    FOREIGN KEY (Country_ID)
    REFERENCES Countries (Country_ID);
```

Nazwa tabeli: Attractions

- Opis: Tabela zawiera informacje o świadczonych atrakcjach dla danej wycieczki.

Nazwa atrybutu	Typ	Opis/Uwagi
Attraction_ID	PK int	Identyfikator atrakcji.
Trip_ID	FK int	Identyfikator wycieczki.
Short_name	FK char(10)	Skrót nazwy atrakcji.
Start_date	date	Data rozpoczęcia atrakcji.
End_date	date	Data zakończenia atrakcji.
Price_p_p	money	Cena atrakcji za jedną osobę.
Quantity	int	Liczba oferowanych miejsc na atrakcję.

- kod DDL

```
-- Table: Attractions
CREATE TABLE Attractions (
    Attraction_ID int IDENTITY(1,1) NOT NULL,
    Trip_ID int  NOT NULL,
    Short_name char(10)  NOT NULL,
    Start_date date  NOT NULL,
    End_date date  NOT NULL,
    Price_p_p money  NOT NULL,
    Quantity int  NOT NULL,
    CONSTRAINT Attractions_pk PRIMARY KEY  (Attraction_ID),
    CONSTRAINT chk_start_date_for_atractions CHECK(Start_date > DATEADD(day, 7, GETDATE())),
    CONSTRAINT chk_end_date_for_atractions CHECK(End_date >= Start_date),
    CONSTRAINT chk_price_for_atractions CHECK(Price_p_p > 0),
    CONSTRAINT chk_quantity_for_atractions CHECK(Quantity > 0)
);
-- Reference: Attractions_Attractions_dictionary (table: Attractions)
ALTER TABLE Attractions ADD CONSTRAINT Attractions_Attractions_dictionary
    FOREIGN KEY (Short_name)
    REFERENCES Attractions_dictionary (Short_name);

-- Reference: Attractions_Trip_offer (table: Attractions)
ALTER TABLE Attractions ADD CONSTRAINT Attractions_Trip_offer
    FOREIGN KEY (Trip_ID)
    REFERENCES Trip_offer (Trip_ID);
```

Nazwa tabeli: Attractions_dictionary

- Opis: Tabela słownikowa zawierająca nazwy atrakcji wraz z opisem.

Nazwa atrybutu	Typ	Opis/Uwagi
Short_name	PK char(10)	Skrót nazwy atrakcji.
Name	varchar(30)	Nazwa atrakcji.
Description	varchar(100)	Opis atrakcji.

- kod DDL

```
-- Table: Attractions_dictionary
CREATE TABLE Attractions_dictionary (
    Short_name char(10)  NOT NULL,
    Name varchar(30)  NOT NULL,
    Description varchar(100)  NOT NULL,
    CONSTRAINT Attractions_dictionary_pk PRIMARY KEY  (Short_name)
);
```

Nazwa tabeli: Reservations

- Opis: Tabela zawiera informacje o rezerwacjach/zamówieniach dokonywanych przez klientów.

Nazwa atrybutu	Typ	Opis/Uwagi
Reservation_ID	PK int	Identyfikator rezerwacji.
Trip_ID	FK int	Identyfikator wycieczki.
Customer_ID	FK int	Identyfikator klienta.
Status	char(1)	Flaga określająca czy rezerwacja jest w trybie edycji, anulowana, czy została zatwierdzona: - Status = 'E', rezerwacja jest edytowanlna (obecna data - 7 dni nie przekracza daty rozpoczęcia wycieczki). - Status = 'A' rezerwacja nie może być edytowana (została przekroczona data rozpoczecia wycieczki - 7 dni, dodatkowo są spełnione warunki: podane zostały dane wszystkich uczestników, zamówienie zostało opłacone). - Status = 'C', w przypadku niespełnienia jakiegokolwiek warunku po czasie (obecna data - 7 dni), zamówenie zostaje anulowane.
Quantity	int	Liczba zarezerwowanych/zamówionych biletów na wycieczkę.
Price_p_p	money	Cena miejsca na wycieczkę za jedną osobę.

- kod DDL

```
-- Table: Reservations
CREATE TABLE Reservations (
    Reservation_ID int IDENTITY(1,1)  NOT NULL,
    Trip_ID int  NOT NULL,
    Customer_ID int  NOT NULL,
    Status char(1)  NOT NULL DEFAULT('E'),
    Quantity int  NOT NULL,
    Price_p_p money  NOT NULL,
    CONSTRAINT Reservations_pk PRIMARY KEY  (Reservation_ID),
    CONSTRAINT chk_quantity_for_reservation CHECK(Quantity > 0),
    CONSTRAINT chk_price_for_reservation CHECK(Price_p_p > 0),
    CONSTRAINT chk_type_reservation CHECK (Status IN ('E', 'A', 'C'))
);
-- Reference: Reservations_Customers (table: Reservations)
ALTER TABLE Reservations ADD CONSTRAINT Reservations_Customers
    FOREIGN KEY (Customer_ID)
    REFERENCES Customers (Customer_ID);

-- Reference: Reservations_Trip_offer (table: Reservations)
ALTER TABLE Reservations ADD CONSTRAINT Reservations_Trip_offer
    FOREIGN KEY (Trip_ID)
    REFERENCES Trip_offer (Trip_ID);
```

Nazwa tabeli: Reservations_details

- Opis: Tabela zawiera szczegóły dotyczące rezerwacji/zamówienia, obejmujące informacje o liczbie uczestników i ich atrakcjach.

Nazwa atrybutu	Typ	Opis/Uwagi
Attraction_ID	PK FK int	Identyfikator atrakcji.
Reservation_ID	PK FK int	Identyfikator rezerwacji.
Quantity	int	Liczba zarezerwowanych/zamówionych biletów na daną atrakcję.
Price_p_p	money	Cena miejsca na atrakcję za jedną osobę.

- kod DDL

```
-- Table: Reservations_details
CREATE TABLE Reservations_details (
    Attraction_ID int  NOT NULL,
    Reservation_ID int  NOT NULL,
    Quantity int  NOT NULL,
    Price_p_p money  NOT NULL,
    CONSTRAINT Reservations_details_pk PRIMARY KEY  (Attraction_ID,Reservation_ID),
    CONSTRAINT chk_quantity_for_reservation_details CHECK(Quantity > 0),
    CONSTRAINT chk_price_for_reservation_details CHECK(Price_p_p > 0)
);
-- Reference: Reservations_details_Attractions (table: Reservations_details)
ALTER TABLE Reservations_details ADD CONSTRAINT Reservations_details_Attractions
    FOREIGN KEY (Attraction_ID)
    REFERENCES Attractions (Attraction_ID);

-- Reference: Reservations_details_Reservations (table: Reservations_details)
ALTER TABLE Reservations_details ADD CONSTRAINT Reservations_details_Reservations
    FOREIGN KEY (Reservation_ID)
    REFERENCES Reservations (Reservation_ID);
```

Nazwa tabeli: Participants

- Opis: Tabela zawiera dane uczestników wycieczki.

Nazwa atrybutu	Typ	Opis/Uwagi
Participant_ID	PK int	Identyfikator uczestnika.
Reservation_ID	FK int	Identyfikator rezerwacji do której należy uczestnik.
First_name	varchar(30)	Imię uczestnika.
Last_name	varchar(30)	Nazwisko uczestnika.
Date_of_birth	date	Data urodzenia uczestnika.

- kod DDL

```
-- Table: Participants
CREATE TABLE Participants (
    Participant_ID int IDENTITY(1,1)  NOT NULL,
    Reservation_ID int  NOT NULL,
    First_name varchar(30)  NOT NULL,
    Last_name varchar(30)  NOT NULL,
    Date_of_birth date  NOT NULL,
    CONSTRAINT Participants_pk PRIMARY KEY  (Participant_ID),
    CONSTRAINT chk_date_of_birth_for_participant
    CHECK(Date_of_birth > '1800-01-01' and Date_of_birth < GETDATE())
);
-- Reference: Participants_Reservations (table: Participants)
ALTER TABLE Participants ADD CONSTRAINT Participants_Reservations
    FOREIGN KEY (Reservation_ID)
    REFERENCES Reservations (Reservation_ID);
```

Nazwa tabeli: Attraction_for_participant

- Opis: Tabela wiążąca uczestnika z odpowiednią atrakcją rezerwacji.

Nazwa atrybutu	Typ	Opis/Uwagi
Participant_ID	PK FK int	Identyfikator uczestnika.
Reservation_ID	PK FK int	Identyfikator rezerwacji do której należy uczestnik.
Attraction_ID	PK FK int	Identyfikator atrakcji którą rezerwuje uczestnik.

- kod DDL

```
-- Table: Attraction_for_participant
CREATE TABLE Attraction_for_participant (
    Participant_ID int  NOT NULL,
    Reservation_ID int  NOT NULL,
    Attraction_ID int  NOT NULL,
    CONSTRAINT Attraction_for_participant_pk PRIMARY KEY  (Participant_ID,Reservation_ID,Attraction_ID)
);
-- Reference: Attraction_for_participant_Participants (table: Attraction_for_participant)
ALTER TABLE Attraction_for_participant ADD CONSTRAINT Attraction_for_participant_Participants
    FOREIGN KEY (Participant_ID)
    REFERENCES Participants (Participant_ID);

-- Reference: Attraction_for_participant_Reservations_details (table: Attraction_for_participant)
ALTER TABLE Attraction_for_participant ADD CONSTRAINT Attraction_for_participant_Reservations_details
    FOREIGN KEY (Attraction_ID,Reservation_ID)
    REFERENCES Reservations_details (Attraction_ID,Reservation_ID);
```

Nazwa tabeli: Company

- Opis: Tabela zawiera informacje na temat nazwy firmy oraz jej numeru NIP.

Nazwa atrybutu	Typ	Opis/Uwagi
NIP	PK int	Numer Identyfikacji Podatkowej (NIP) firmy.
Company_name	varchar(30)	Nazwa firmy.

- kod DDL

```
-- Table: Company
CREATE TABLE Company (
    NIP int  NOT NULL,
    Company_name varchar(30)  NOT NULL,
    CONSTRAINT Company_pk PRIMARY KEY  (NIP)
);
```

Nazwa tabeli: Customers

- Opis: Tabela zawiera dane klientów indywidualnych oraz firmowych, którzy są właścicielami zamówień.

Nazwa atrybutu	Typ	Opis/Uwagi
Customer_ID	PK int	Identyfikator klienta.
City_ID	FK int	Identyfikator kraju siedziby firmy lub adres zamieszkania osoby prywatnej.
Street	varchar(50)	Ulica i numer budynku zamieszkania klienta lub ulica i numer budynku siedziby firmy.
First_name	varchar(30)	Imię klienta.
Last_name	varchar(30)	Nazwisko klienta.
Date_of_birth	date	Data urodzenia klienta.
NIP	FK int	Numer Identyfikacji Podatkowej (NIP) firmy klienta, jeśli klient jest osobą indywidualną NIP = NULL.
Phone	varchar(20)	Numer telefonu klienta.
Email	varchar(30)	Adres e-mail klienta.

- kod DDL

```
-- Table: Customers
CREATE TABLE Customers (
    Customer_ID int IDENTITY(1,1) NOT NULL,
    City_ID int  NOT NULL,
    Street varchar(50)  NOT NULL,
    First_name varchar(30)  NOT NULL,
    Last_name varchar(30)  NOT NULL,
    Date_of_birth date  NOT NULL,
    NIP int  NULL,
    Phone varchar(20) NOT NULL,
    Email varchar(30) UNIQUE NOT NULL,
    CONSTRAINT Customers_pk PRIMARY KEY  (Customer_ID),
    CONSTRAINT chk_date_of_birth_for_customer
    CHECK(Date_of_birth > '1800-01-01' AND Date_of_birth < GETDATE())
);
-- Reference: Customers_Company (table: Customers)
ALTER TABLE Customers ADD CONSTRAINT Customers_Company
    FOREIGN KEY (NIP)
    REFERENCES Company (NIP);

-- Reference: Customers_Cities (table: Customers)
ALTER TABLE Customers ADD CONSTRAINT Customers_Cities
    FOREIGN KEY (City_ID)
    REFERENCES Cities (City_ID);
```

Nazwa tabeli: Financial_operations

- Opis: Tabela zawiera operacje finansowe, czy mające charakter wpłaty, czy też wypłaty.

Nazwa atrybutu	Typ	Opis/Uwagi
Operation_ID	PK int	Identyfikator operacji.
Reservation_ID	FK int	Identyfikator rezerwacji, na którą klient wpłaca środki lub z której środki są wypłacane.
Date	datetime	Czas wykonania operacji.
Amount	money	Kwota operacji w złotówkach (dodatnia kwota oznacza wpłatę, ujemna kwota oznacza wypłatę).
Type	char(1)	Flaga określająca sposób płatności: -Type = "C" (Credit Card) Płatność wykonana za pomocą karty płatniczej. -Type = "M" (Money) Płatność wykonana za pomocą gotówki. -Type = "T" (Transfer) Płatność wykonana za pomocą przelewu bankowego.

- kod DDL

```
-- Table: Financial_operations
CREATE TABLE Financial_operations (
    Operation_ID int IDENTITY(1,1)  NOT NULL,
    Reservation_ID int  NOT NULL,
    Date datetime  NOT NULL DEFAULT(GETDATE()),
    Amount money  NOT NULL,
    Type char(1)  NOT NULL,
    CONSTRAINT Financial_operations_pk PRIMARY KEY  (Operation_ID),
    CONSTRAINT chk_amount CHECK(Amount != 0),
    CONSTRAINT chk_type CHECK (Type IN ('C', 'M', 'T'))
);
-- Reference: Financial_operations_Reservations (table: Financial_operations)
ALTER TABLE Financial_operations ADD CONSTRAINT Financial_operations_Reservations
    FOREIGN KEY (Reservation_ID)
    REFERENCES Reservations (Reservation_ID);
```

3. Widoki, procedury/funkcje, triggery

Widoki

Kody poleceń definiujących dostępne widoki wraz z komentarzami.

Wyświetlenie ofert dostępnych wycieczek

```
CREATE VIEW Available_trips AS
SELECT
    T.Trip_ID,
    Co.Country,
    C.City,
    T.Start_date,
    T.End_date,
    T.Quantity,
    T.Price_p_p
FROM
    Trip_offer T
JOIN
    Cities C on C.City_ID = T.City_ID
JOIN Countries Co on C.Country_ID = Co.Country_ID
WHERE
    Share_date <= GETDATE() AND
    Start_date >= DATEADD(day, 7, GETDATE());
```

Komentarz

Powyższy widok wyświetla wszystkie dostępne wycieczki, to znaczy takie, które zostały już opublikowane oraz zaczynają się minimum za 7 dni.

Wynik widoku (pierwsze 4 wyniki)

Trip_ID	Country	City	Start_date	End_date	Quantity	Price_p_p
1	Polska	Kraków	2024-08-24	2024-08-29	10	500,00
2	Wielka Brytania	Londyn	2024-08-24	2024-08-29	10	500,00
3	Stany Zjednoczone	Nowy Jork	2024-08-24	2024-08-29	10	500,00
4	Francja	Paryz	2024-08-24	2024-08-29	10	500,00

Wyświetlenie ofert dostępnych wycieczek z ilością miejsc

```
CREATE VIEW Available_trips_quantity AS
SELECT
    T.Trip_ID,
    Co.Country,
    C.City,
    T.Start_date,
    T.End_date,
    T.Price_p_p,
    T.Quantity - ISNULL(SUM(R.Quantity), 0) AS Available_Quantity
FROM
    Trip_offer T
JOIN
    Cities C ON C.City_ID = T.City_ID
JOIN
    Countries Co ON C.Country_ID = Co.Country_ID
LEFT JOIN
    Reservations R ON T.Trip_ID = R.Trip_ID AND R.Status = 'E'
WHERE
    T.Share_date <= GETDATE() AND
    T.Start_date >= DATEADD(day, 7, GETDATE())
GROUP BY
    T.Trip_ID,
    Co.Country,
    C.City,
    T.Start_date,
    T.End_date,
    T.Price_p_p,
    T.Quantity
HAVING
    T.Quantity - ISNULL(SUM(R.Quantity), 0) > 0;
```

Komentarz

Powyższy widok wyświetla wszystkie dostępne wycieczki wraz z liczbą pozostałych wolnych miejsc, co jest obliczane za pomocą tabeli Reservations.

Wynik widoku (pierwsze 4 wyniki)

Trip_ID	Country	City	Start_date	End_date	Price_p_p	Available_Quantity
1	Polska	Kraków	2024-08-24	2024-08-29	500,00	10
2	Wielka Brytania	Londyn	2024-08-24	2024-08-29	500,00	1
3	Stany Zjednoczone	Nowy Jork	2024-08-24	2024-08-29	500,00	10
5	Japonia	Tokio	2024-08-24	2024-08-29	500,00	4

Wyświetlenie ofert dostępnych atrakcji

```
CREATE VIEW Available_attractions AS
SELECT
    T.Trip_ID,
    A.Attraction_ID,
    C.City,
    T.Start_date,
    T.End_date,
    Ad.Name AS Attraction_name,
    A.Price_p_p
FROM
    Attractions A
JOIN
    Trip_offer T on A.Trip_ID = T.Trip_ID
JOIN
    Cities C on T.City_ID = C.City_ID
JOIN
    Attractions_dictionary Ad on A.Short_name = Ad.Short_name
WHERE
    T.Share_date <= GETDATE() AND
    T.Start_date >= DATEADD(day, 7, GETDATE());
```

Komentarz

Powyższy widok wyświetla wszystkie dostępne atrakcje, czyli ze sprawdzeniem czy wycieczki do których należą zostały już opublikowane oraz zaczynają się minimum za 7 dni.

Wynik widoku (pierwsze 4 wyniki)

Trip_ID	Attraction_ID	City	Start_date	End_date	Attraction_name	Price_p_p
2	1	Londyn	2024-08-24	2024-08-29	Rynek Główny w Krakowie	40,00
2	2	Londyn	2024-08-24	2024-08-29	Zamek Wawel w Krakowie	55,00
3	3	Nowy Jork	2024-08-24	2024-08-29	British Museum w Londynie	100,00
3	4	Nowy Jork	2024-08-24	2024-08-29	Tower Bridge w Londynie	60,00

Wyświetlenie ofert dostępnych atrakcji z ilością miejsc

```
CREATE VIEW Available_attractions_quantity AS
WITH TempTable AS (
    SELECT
        T.Trip_ID,
        A.Attraction_ID,
        R.Reservation_ID,
        C.City,
        A.Start_date,
        A.End_date,
        A.Price_p_p,
        A.Quantity,
        Res.Status,
        ISNULL(SUM(R.Quantity), 0) AS Sum_Quantity
    FROM
        Attractions A
    JOIN
        Trip_offer T ON A.Trip_ID = T.Trip_ID
    JOIN
        Cities C ON T.City_ID = C.City_ID
    LEFT JOIN
        Reservations_details R ON A.Attraction_ID = R.Attraction_ID
    LEFT JOIN
        Reservations Res ON R.Reservation_ID = Res.Reservation_ID AND Res.Status = 'E'
    WHERE
        T.Share_date <= GETDATE() AND
        T.Start_date >= DATEADD(day, 7, GETDATE())
    GROUP BY
        T.Trip_ID,
        A.Attraction_ID,
        R.Reservation_ID,
        C.City,
        A.Start_date,
        A.End_date,
        A.Price_p_p,
        A.Quantity,
        Res.Status
)

SELECT
    Trip_ID,
    Attraction_ID,
    City,
    Start_date,
    End_date,
    Price_p_p,
    MAX(Quantity) - SUM(Sum_Quantity) AS Quantity_Difference
FROM
    TempTable
GROUP BY
    Trip_ID,
    Attraction_ID,
    City,
    Start_date,
    End_date,
    Price_p_p;
```

Komentarz

Powyższy widok wyświetla wszystkie dostępne atrakcje wraz z liczbą pozostałych wolnych miejsc, co jest obliczane za pomocą tabeli Reservations_details.

Wynik widoku (pierwsze 4 wyniki)

Trip_ID	Attraction_ID	City	Start_date	End_date	Price_p_p	Quantity_Difference
2	1	Londyn	2024-08-25	2024-08-25	60,00	6
2	2	Londyn	2024-08-27	2024-08-27	55,00	7
3	3	Nowy Jork	2024-08-25	2024-08-25	100,00	5
3	4	Nowy Jork	2024-08-27	2024-08-27	60,00	3

Wyświetlanie raportu stanu opłaty każdej rezerwacji każdego klienta

```
CREATE VIEW Client_reservation_saldo AS
WITH Reservation_cost AS
(
    SELECT
        C.Customer_ID,
        C.First_name,
        C.Last_name,
        C.NIP,
        Com.Company_name,
        R.Reservation_ID,
        ISNULL(R.Price_p_p * R.Quantity, 0) AS Cost_T,
        ISNULL(SUM(F.Amount), 0) AS SUM_Amount
    FROM
        Customers C
    LEFT JOIN
        Reservations R ON C.Customer_ID = R.Customer_ID AND R.Status = 'E'
    LEFT JOIN
        Financial_operations F ON R.Reservation_ID = F.Reservation_ID
    LEFT JOIN
        Company Com ON C.NIP = Com.NIP
    GROUP BY
        C.Customer_ID,
        C.First_name,
        C.Last_name,
        C.NIP,
        Com.Company_name,
        R.Reservation_ID,
        R.Price_p_p,
        R.Quantity
),
Attraction_cost AS
(
    SELECT
        Customer_ID,
        First_name,
        Last_name,
        NIP,
        Company_name,
        Reservation_ID,
        SUM(COST_A) AS COST_A_C
    FROM (
        SELECT
            C.Customer_ID,
            C.First_name,
            C.Last_name,
            C.NIP,
            Com.Company_name,
            RD.Reservation_ID,
            ISNULL(RD.Price_p_p * RD.Quantity, 0) AS COST_A
        FROM
            Customers C
        LEFT JOIN
            Reservations R ON C.Customer_ID = R.Customer_ID AND R.Status = 'E'
        LEFT JOIN
            Reservations_details RD ON R.Reservation_ID = RD.Reservation_ID
        LEFT JOIN
            Company Com ON C.NIP = Com.NIP
        GROUP BY
            C.Customer_ID,
            C.First_name,
            C.Last_name,
            C.NIP,
            Com.Company_name,
            RD.Reservation_ID,
            RD.Price_p_p,
```

```
            RD.Quantity
    ) AS tempAttraction_cost
    GROUP BY
        Customer_ID,
        First_name,
        Last_name,
        NIP,
        Company_name,
        Reservation_ID
)
SELECT
    RC.Customer_ID,
    RC.First_name,
    RC.Last_name,
    RC.NIP,
    RC.Company_name,
    RC.Reservation_ID,
    (SUM_Amount-Cost_T-ISNULL(COST_A_C,0)) AS SALDO
FROM
    Reservation_cost RC
LEFT JOIN
    Attraction_cost AC on RC.Reservation_ID = AC.Reservation_ID
```

Komentarz

Powyższy widok wyświetla wszystkie rezerwacje znajdujące się w stanie edycji wraz z danymi osoby dokonującej rezerwacji oraz zsumowane saldo rezerwacji (dotychczasowe wpłaty/wypłaty - całkowity koszt wycieczki i atrakcji).

Wynik widoku (pierwsze 4 wyniki)

Customer_ID	First_name	Last_name	NIP	Company_name	Reservation_ID	SALDO
1	Anna	Kowalska	123456789	Tech Solutions Sp. z o.o.	1	0,00
2	Jan	Nowak	NULL	NULL	2	5740,00
3	Marta	Wisniewska	987654321	Zielona Energia S.A.	3	-3075,00
4	Piotr	Kowalczyk	NULL	NULL	NULL	0,00

Wyświetlanie raportu stanu opłaty każdego klienta dla edytowalnych rezerwacji

```
CREATE VIEW Client_saldo AS
WITH Reservation_cost AS
(
    SELECT
        C.Customer_ID,
        C.First_name,
        C.Last_name,
        C.NIP,
        Com.Company_name,
        R.Reservation_ID,
        ISNULL(R.Price_p_p * R.Quantity, 0) AS Cost_T,
        ISNULL(SUM(F.Amount), 0) AS SUM_Amount
    FROM
        Customers C
    LEFT JOIN
        Reservations R ON C.Customer_ID = R.Customer_ID AND R.Status = 'E'
    LEFT JOIN
        Financial_operations F ON R.Reservation_ID = F.Reservation_ID
    LEFT JOIN
        Company Com ON C.NIP = Com.NIP
    GROUP BY
        C.Customer_ID,
        C.First_name,
        C.Last_name,
        C.NIP,
        Com.Company_name,
        R.Reservation_ID,
        R.Price_p_p,
        R.Quantity
),
Attraction_cost AS
(
    SELECT
        Customer_ID,
        First_name,
        Last_name,
        NIP,
        Company_name,
        Reservation_ID,
        SUM(COST_A) AS COST_A_C
    FROM (
        SELECT
            C.Customer_ID,
            C.First_name,
            C.Last_name,
            C.NIP,
            Com.Company_name,
            RD.Reservation_ID,
            ISNULL(RD.Price_p_p * RD.Quantity, 0) AS COST_A
        FROM
            Customers C
        LEFT JOIN
            Reservations R ON C.Customer_ID = R.Customer_ID AND R.Status = 'E'
        LEFT JOIN
            Reservations_details RD ON R.Reservation_ID = RD.Reservation_ID
        LEFT JOIN
            Company Com ON C.NIP = Com.NIP
        GROUP BY
            C.Customer_ID,
            C.First_name,
            C.Last_name,
            C.NIP,
            Com.Company_name,
            RD.Reservation_ID,
            RD.Price_p_p,
```

```
            RD.Quantity
    ) AS tempAttraction_cost
    GROUP BY
        Customer_ID,
        First_name,
        Last_name,
        NIP,
        Company_name,
        Reservation_ID
)
SELECT
    RC.Customer_ID,
    RC.First_name,
    RC.Last_name,
    RC.NIP,
    RC.Company_name,
    SUM((SUM_Amount-Cost_T-ISNULL(COST_A_C,0))) AS SALDO
FROM
    Reservation_cost RC
LEFT JOIN
    Attraction_cost AC on RC.Reservation_ID = AC.Reservation_ID
GROUP BY
    RC.Customer_ID,
    RC.First_name,
    RC.Last_name,
    RC.NIP,
    RC.Company_name
```

Komentarz

Powyższy widok wyświetla listę klientów oraz ich sumaryczne salda za wszystkie obecne zamówienia/rezerwacje (czyli te, które są jeszcze dostępne do edycji).

Wynik widoku (pierwsze 4 wyniki)

Customer_ID	First_name	Last_name	NIP	Company_name	SALDO
1	Anna	Kowalska	123456789	Tech Solutions Sp. z o.o.	0,00
2	Jan	Nowak	NULL	NULL	5740,00
3	Marta	Wisniewska	987654321	Zielona Energia S.A.	-3075,00
4	Piotr	Kowalczyk	NULL	NULL	0,00

Wyświetlanie raportu stanu opłaty każdego klienta dla historycznych rezerwacji

```
CREATE VIEW Client_saldo_hist AS
WITH Reservation_cost AS
(
    SELECT
        C.Customer_ID,
        C.First_name,
        C.Last_name,
        C.NIP,
        Com.Company_name,
        R.Reservation_ID,
        ISNULL(R.Price_p_p * R.Quantity, 0) AS Cost_T,
        ISNULL(SUM(F.Amount), 0) AS SUM_Amount
    FROM
        Customers C
    LEFT JOIN
        Reservations R ON C.Customer_ID = R.Customer_ID AND (R.Status = 'A' OR R.Status = 'C')
    LEFT JOIN
        Financial_operations F ON R.Reservation_ID = F.Reservation_ID
    LEFT JOIN
            Company Com ON C.NIP = Com.NIP
    GROUP BY
        C.Customer_ID,
        C.First_name,
        C.Last_name,
        C.NIP,
        Com.Company_name,
        R.Reservation_ID,
        R.Price_p_p,
        R.Quantity
),
Attraction_cost AS
(
    SELECT
        Customer_ID,
        First_name,
        Last_name,
        NIP,
        Company_name,
        Reservation_ID,
        SUM(COST_A) AS COST_A_C
    FROM (
        SELECT
            C.Customer_ID,
            C.First_name,
            C.Last_name,
            C.NIP,
            Com.Company_name,
            RD.Reservation_ID,
            ISNULL(RD.Price_p_p * RD.Quantity, 0) AS COST_A
        FROM
            Customers C
        LEFT JOIN
            Reservations R ON C.Customer_ID = R.Customer_ID AND (R.Status = 'A' OR R.Status = 'C')
        LEFT JOIN
            Reservations_details RD ON R.Reservation_ID = RD.Reservation_ID
        LEFT JOIN
            Company Com ON C.NIP = Com.NIP
        GROUP BY
            C.Customer_ID,
            C.First_name,
            C.Last_name,
            C.NIP,
            Com.Company_name,
            RD.Reservation_ID,
            RD.Price_p_p,
```

```
            RD.Quantity
    ) AS tempAttraction_cost
    GROUP BY
        Customer_ID,
        First_name,
        Last_name,
        NIP,
        Company_name,
        Reservation_ID
)
SELECT
    RC.Customer_ID,
    RC.First_name,
    RC.Last_name,
    RC.NIP,
    RC.Company_name,
    SUM((SUM_Amount-Cost_T-ISNULL(COST_A_C,0))) AS SALDO
FROM
    Reservation_cost RC
LEFT JOIN
    Attraction_cost AC on RC.Reservation_ID = AC.Reservation_ID
GROUP BY
    RC.Customer_ID,
    RC.First_name,
    RC.Last_name,
    RC.NIP,
    RC.Company_name
```

Komentarz

Powyższy widok wyświetla listę klientów oraz ich sumaryczne salda za przeszłe zamówienia/rezerwacje (czyli te, które zostały już zaakceptowane lub anulowane).

Wynik widoku (pierwsze 4 wyniki)

Customer_ID	First_name	Last_name	NIP	Company_name	SALDO
6	Michal	Nowicki	NULL	NULL	-4000,00
7	Katarzyna	Kaminska	654321987	Global Trade Sp. z o.o.	0,00
8	Jakob	Dniano	NULL	NULL	0,00
1	Anna	Kowalska	123456789	Tech Solutions Sp. z o.o.	0,00

Wyświetlanie wszystkich uczestników akceptowanych rezerwacji z atrakcjami

```
CREATE VIEW All_accepted_participants AS
    SELECT
        P.First_name,
        P.Last_name,
        C.City,
        T.Start_date,
        ISNULL(Ad.Name,'Brak atrakcji') as Atrakcja,
        T.Trip_ID,
        A.Attraction_ID
    FROM
        Participants P
    LEFT JOIN
        Attraction_for_participant AP on P.Participant_ID = AP.Participant_ID
    LEFT JOIN
        Reservations_details RD on AP.Attraction_ID = RD.Attraction_ID
        and AP.Reservation_ID = RD.Reservation_ID
    LEFT JOIN
        Attractions A on RD.Attraction_ID = A.Attraction_ID
    JOIN
        Reservations R on P.Reservation_ID = R.Reservation_ID and R.Status = 'A'
    LEFT JOIN
        Trip_offer T on R.Trip_ID = T.Trip_ID
    JOIN
        Cities C on T.City_ID = C.City_ID
    JOIN
        Attractions_dictionary Ad on A.Short_name = Ad.Short_name
```

Komentarz

Powyższy widok wyświetla wszystkich uczestników zaakceptowanych wycieczek, wraz z nazwą miasta do którego jest wycieczka, datą rozpoczęcia oraz spisem atrakcji.

Wynik widoku (pierwsze 4 wyniki)

First_name	Last_name	City	Start_date	Atrakcja	Trip_ID	Attraction_ID
Katarzyna	Nowak	Madryt	2023-07-01	Plaza Mayor	7	18
Marek	Kowalski	Madryt	2023-07-01	Plaza Mayor	7	18
Ewa	Wisniewska	Madryt	2023-07-01	Plaza Mayor	7	18
Jan	Wójcik	Madryt	2023-07-01	Plaza Mayor	7	18

Wyświetlanie rezerwacji które zostały akceptowane

```
CREATE VIEW Customer_accepted_reservation AS
    SELECT
        C.First_name,
        C.Last_name,
        C.NIP,
        Com.Company_name,
        R.Reservation_ID,
        R.Quantity,
        Ci.City,
        T.Start_date
    FROM
        Reservations R
    JOIN
        Trip_offer T ON R.Trip_ID = T.Trip_ID
    JOIN
        Customers C ON R.Customer_ID = C.Customer_ID
    JOIN
        Cities Ci ON T.City_ID = Ci.City_ID
    LEFT JOIN
        Company Com ON C.NIP = Com.NIP
    WHERE
        R.Status = 'A'
```

Komentarz

Powyższy widok wyświetla spis zaakceptowanych rezerwacji.

Wynik widoku (pierwszy wynik)

First_name	Last_name	NIP	Company_name	Reservation_ID	Quantity	City	Start_date
Aleksandra	Lis	NULL	NULL	6	20	Madryt	2023-07-01

Wyświetlanie raportu operacji finansowych.

```
CREATE VIEW Financial_report AS
    SELECT
        C.Customer_ID,
        C.First_name,
        C.Last_name,
        C.NIP,
        Com.Company_name,
        F.Operation_ID,
        F.Date,
        F.Amount,
        F.Reservation_ID
    FROM
        Financial_operations F
    JOIN
        Reservations R on F.Reservation_ID = R.Reservation_ID
    JOIN
        Customers C on R.Customer_ID = C.Customer_ID
    LEFT JOIN
        Company Com ON C.NIP = Com.NIP
```

Komentarz

Powyższy widok wyświetla raport wpływów i wypływów środków.

Wynik widoku (pierwsze 4 wyniki)

Customer_ID	First_name	Last_name	NIP	Company_name	Operation_ID	Date	Amount	Reservation_ID
6	Michal	Nowicki	NULL	NULL	1	2024-05-16 11:53:47.750	500,00	4
2	Jan	Nowak	NULL	NULL	5	2024-05-16 11:53:47.750	10000,00	2
2	Jan	Nowak	NULL	NULL	7	2024-05-16 11:53:47.750	-100,00	2
1	Anna	Kowalska	123456789	Tech Solutions Sp. z o.o.	12	2024-05-19 19:45:20.290	3300,00	1

Wyświetlanie uczestnitków wszystkich wycieczek z podziałem na dzieci i dorosłych

```
CREATE VIEW Participants_By_Age AS
SELECT
    p.Participant_ID,
    p.First_name,
    p.Last_name,
    p.Date_of_birth,
    CASE
        WHEN DATEDIFF(year, p.Date_of_birth, GETDATE()) >= 18 THEN 'Adult'
        ELSE 'Child'
    END AS Age_Group
FROM
    Participants P;
```

Komentarz

Powyższy widok wyświetla spis wszystkich zarejestrowanych w systemie uczestników z podziałem na dorosłych oraz dzieci.

Wynik widoku (pierwsze 4 wyniki)

Participant_ID	First_name	Last_name	Date_of_birth	Age_Group
4	Antonina	Nowak	1971-11-12	Adult
5	Maria	Szkielet-Zaporowska	1968-02-12	Adult
6	Zygmunt	Flisiuk	1974-09-21	Adult
7	Janusz	Ancymon	1977-01-01	Adult

Wyświetlanie czy rezerwacje zostały opłacone oraz zostali podani uczestnicy

```
CREATE VIEW Reservations_condition AS
    WITH Sum_Reservation AS (
        SELECT
            R.Reservation_ID,
            COUNT(*)-MAX(R.Quantity) AS Reservation_diff
        FROM
            Reservations R
        LEFT JOIN
            Participants P ON R.Reservation_ID = P.Reservation_ID
        WHERE
            R.Status = 'E'
        GROUP BY
            R.Reservation_ID
    ),
    Sum_Attraction AS (
        SELECT
            RD.Reservation_ID,
            RD.Attraction_ID,
            count(*)-MAX(RD.Quantity) as Attraction_diff
        FROM Reservations_details RD
        LEFT JOIN
        Attraction_for_participant AP on RD.Attraction_ID = AP.Attraction_ID
        AND RD.Reservation_ID = AP.Reservation_ID
        LEFT JOIN
        Participants P on AP.Participant_ID = P.Participant_ID
        GROUP BY
            RD.Reservation_ID,
            RD.Attraction_ID
    )
    SELECT
        SR.Reservation_ID,
        SA.Attraction_ID,
        CASE WHEN SR.Reservation_diff = 0 THEN 'OK' ELSE 'ERROR' END AS Participants_for_trip,
        CASE WHEN SA.Attraction_diff = 0 THEN 'OK' ELSE 'ERROR' END AS Participants_for_Attraction,
        CASE WHEN CS.SALDO = 0.00 THEN 'OK' ELSE 'ERROR' END AS Paid
    FROM
        Sum_Reservation SR
    LEFT JOIN
        Sum_Attraction SA ON SR.Reservation_ID = SA.Reservation_ID
    JOIN
        Client_reservation_saldo CS on SR.Reservation_ID = CS.Reservation_ID
```

Komentarz

Powyższy widok wyświetla wszystkie rezerwacje w stanie edycji wraz z informacją o stanie elementów wymaganych do akceptacji: czy podani zostali uczestnicy wycieczki oraz poszczególnych atrakcji oraz czy rezerwacja została opłacona. Jeśli tabela wyświetla trzy razy 'OK' przy numerze rezerwacji, oznacza to, że dana rezerwacja może zostać zaakceptowana.

Wynik widoku (pierwsze 4 wyniki)

Reservation_ID	Attraction_ID	Participants_for_trip	Participants_for_Attraction	Paid
1	5	OK	OK	OK
2	5	ERROR	ERROR	ERROR
2	6	ERROR	ERROR	ERROR
3	8	ERROR	OK	ERROR

Procedury/funkcje

Spis procedur oraz funkcji wraz z kodem DLL oraz komentarzem.

Procedury

Procedura tworzenia nowej wycieczki.

```
CREATE PROCEDURE InsertTripOffer
    @City_Name NVARCHAR(100),
    @Price_p_p MONEY,
    @Start_date DATE,
    @End_date DATE,
    @Quantity INT,
    @Share_date DATETIME
AS
BEGIN
    DECLARE @City_ID INT;
    DECLARE @Trip_ID INT;

    -- Get City_ID from Cities table
    SELECT @City_ID = City_ID FROM Cities WHERE City = @City_Name;

    IF @City_ID IS NULL
    BEGIN
        RAISERROR('City not found', 16, 1);
        RETURN;
    END

    -- Insert new trip
    INSERT INTO Trip_offer (City_ID, Start_date, End_date, Quantity, Price_p_p, Share_date)
    VALUES (@City_ID, @Start_date, @End_date, @Quantity, @Price_p_p, @Share_date);

    -- Get the ID of the newly inserted trip
    SELECT @Trip_ID = SCOPE_IDENTITY();

    SELECT * FROM Trip_offer WHERE Trip_ID = @Trip_ID;
END;
GO
```

Komentarz

Procedura pozwala dodawać nową wycieczkę do tabeli Trip_offer. Procedura kończy się wyświetleniem jaki wpis został utworzony.

Przykładowe wywołanie procedury

```
EXEC InsertTripOffer
    @City_Name='Berlin',
    @Price_p_p=2500,
    @Start_date='2026-07-01',
    @End_date='2026-07-10',
    @Quantity=15,
    @Share_date='2024-06-03';
```

Trip_ID	City_ID	Start_date	End_date	Quantity	Price_p_p	Share_date
11	7	2026-07-01	2026-07-10	15	2500,00	2024-06-03 00:00:00.000

Procedura uaktualnienia wycieczki.

```
CREATE PROCEDURE UpdateTripOffer
    @Trip_ID INT,
    @Price_p_p MONEY = NULL,
    @Quantity INT = NULL
AS
BEGIN
    DECLARE @CurrentQuantity INT;
    DECLARE @AvailableQuantity INT;
    DECLARE @StartDate DATE;

    IF @Trip_ID IS NULL
    BEGIN
        RAISERROR('Trip_ID is required for update', 16, 1);
        RETURN;
    END

    -- Get the current quantity and start date from Trip_offer
    SELECT @CurrentQuantity = Quantity, @StartDate = Start_date FROM Trip_offer WHERE Trip_ID = @Trip_ID;

    IF @CurrentQuantity IS NULL
    BEGIN
        RAISERROR('Trip not found', 16, 1);
        RETURN;
    END

    -- Get the available quantity from the Available_trips_quantity view
    SELECT @AvailableQuantity = Available_Quantity FROM Available_trips_quantity WHERE Trip_ID = @Trip_ID;

    IF @AvailableQuantity IS NULL
    BEGIN
        RAISERROR('Trip not found in Available_trips_quantity view', 16, 1);
        RETURN;
    END

    -- Ensure the new quantity is not less than the number of already reserved spots
    IF @Quantity IS NOT NULL AND @Quantity < (@CurrentQuantity - @AvailableQuantity)
    BEGIN
        RAISERROR('Quantity cannot be decreased below the number of already reserved spots', 16, 1);
        RETURN;
    END

    -- Update trip details
    UPDATE Trip_offer
    SET Price_p_p = ISNULL(@Price_p_p, Price_p_p),
        Quantity = ISNULL(@Quantity, Quantity)
    WHERE Trip_ID = @Trip_ID;

    SELECT * FROM Trip_offer WHERE Trip_ID = @Trip_ID;
END;
GO
```

Komentarz

Procedura pozwala zmieniać utworzoną wycieczkę w tabeli Trip_offer. Warunki, które muszą być spełnione:

- Dana wycieczka o Trip_offer musi istnieć
- Nie można zmniejszyć miejsc na wycieczek, jeżeli te miejsca są już zarezerwowane Procedura kończy się wyświetleniem jaki wpis został zmieniony.

Przykładowe wywołanie procedury

```
EXEC UpdateTripOffer
    @Trip_ID = 11,
    @Quantity = 10;
```

Trip_ID	City_ID	Start_date	End_date	Quantity	Price_p_p	Share_date
11	7	2026-07-01	2026-07-10	10	2500,00	2024-06-03 00:00:00.000

Procedura tworzenia nowej atrakcji do danej wycieczki.

```
CREATE PROCEDURE InsertAttraction
    @Trip_ID INT,
    @Short_name CHAR(10),
    @Start_date DATE,
    @End_date DATE,
    @Price_p_p MONEY,
    @Quantity INT
AS
BEGIN
    -- Insert new attraction
    INSERT INTO Attractions (Trip_ID, Short_name, Start_date, End_date, Price_p_p, Quantity)
    VALUES (@Trip_ID, @Short_name, @Start_date, @End_date, @Price_p_p, @Quantity);

    -- Get the ID of the newly inserted attraction
    DECLARE @NewAttraction_ID INT = SCOPE_IDENTITY();

    PRINT 'New attraction added:';
    SELECT * FROM Attractions WHERE Attraction_ID = @NewAttraction_ID;
END;
GO
```

Komentarz

Procedura pozwala dodawać nową atrakcję do danej wycieczki (tabela Attraction). Procedura kończy się wyświetleniem jaki wpis został utworzony.

Przykładowe wywołanie procedury

```
EXEC InsertAttraction
    @Trip_ID = 11,
    @Short_name = 'ZwBelwWied',
    @Start_date = '2026-07-01',
    @End_date = '2026-07-01',
    @Price_p_p = 360,
    @Quantity = 5;
```

Attraction_ID	Trip_ID	Short_name	Start_date	End_date	Price_p_p	Quantity
26	11	ZwBelwWied	2026-07-01	2026-07-01	360,00	5

Procedura aktualizowania atrakcji danej wycieczki.

```
CREATE PROCEDURE UpdateAttraction
    @Attraction_ID INT,
    @Trip_ID INT,
    @Price_p_p MONEY = NULL,
    @Quantity INT = NULL
AS
BEGIN
    DECLARE @CurrentQuantity INT;
    DECLARE @AvailableQuantity INT;

    IF @Attraction_ID IS NULL
    BEGIN
        RAISERROR('Attraction_ID is required for update', 16, 1);
        RETURN;
    END

    IF @Trip_ID IS NULL
    BEGIN
        RAISERROR('Trip_ID is required for update', 16, 1);
        RETURN;
    END

    -- Get the current quantity from Attractions
    SELECT @CurrentQuantity = Quantity FROM Attractions WHERE Attraction_ID = @Attraction_ID;

    IF @CurrentQuantity IS NULL
    BEGIN
        RAISERROR('Attraction not found', 16, 1);
        RETURN;
    END

    -- Get the available quantity from the Available_attractions_quantity view
    SELECT @AvailableQuantity = Quantity_Difference FROM Available_attractions_quantity
    WHERE Trip_ID = @Trip_ID AND Attraction_ID = @Attraction_ID;

    IF @AvailableQuantity IS NULL
    BEGIN
        RAISERROR('Attraction for trip not found in Available_attractions_quantity view', 16, 1);
        RETURN;
    END

    -- Ensure the new quantity is not less than the number of already reserved spots
    IF @Quantity IS NOT NULL AND @Quantity < (@CurrentQuantity - @AvailableQuantity)
    BEGIN
        RAISERROR('Quantity cannot be decreased below the number of already reserved spots', 16, 1);
        RETURN;
    END

    -- Update attraction details
    UPDATE Attractions
    SET Price_p_p = ISNULL(@Price_p_p, Price_p_p),
        Quantity = ISNULL(@Quantity, Quantity)
    WHERE Attraction_ID = @Attraction_ID;

    PRINT 'Attraction updated:';
    SELECT * FROM Attractions WHERE Attraction_ID = @Attraction_ID;
END;
GO
```

Komentarz

Procedura pozwala aktualizować daną atrakcję do danej wycieczki (tabela Attraction). Procedura kończy się wyświetleniem jaki wpis został zmieniony.

Przykładowe wywołanie procedury

```
EXEC UpdateAttraction
    @Attraction_ID = 26,
    @Trip_ID = 11,
    @Quantity = 3;
```

Attraction_ID	Trip_ID	Short_name	Start_date	End_date	Price_p_p	Quantity
26	11	ZwBelwWied	2026-07-01	2026-07-01	360,00	3

Procedura rejestracji nowego użytkownika

```
CREATE PROCEDURE RegisterCustomer
    @City_ID int,
    @Street varchar(50),
    @First_name varchar(30),
    @Last_name varchar(30),
    @Date_of_birth date,
    @NIP int = NULL,
    @Company_name varchar(30) = NULL,
    @Phone varchar(20),
    @Email varchar(30)
AS
BEGIN
    -- Check if the customer is business or individual
    IF @NIP IS NOT NULL
    BEGIN
        -- Business customer, check if the company already exists
        IF NOT EXISTS (SELECT 1 FROM Company WHERE NIP = @NIP)
        BEGIN
            -- Company does not exist, insert new company
            INSERT INTO Company (NIP, Company_name)
            VALUES (@NIP, @Company_name)
        END
    END

    -- Insert new customer into Customers table
    INSERT INTO Customers (City_ID, Street, First_name, Last_name, Date_of_birth, NIP, Phone, Email)
    VALUES (@City_ID, @Street, @First_name, @Last_name, @Date_of_birth, @NIP, @Phone, @Email)

    -- Select the newly created customer using SCOPE_IDENTITY()
    SELECT * FROM Customers WHERE Customer_ID = SCOPE_IDENTITY()
END
```

Komentarz

Procedura pozwala zarejestrować nowego użytkownika w serwisie.

Przykładowe wywołanie procedury

```
EXEC RegisterCustomer
    @City_ID = 2,
    @Street = 'Corporate Avenue',
    @First_name = 'Jane',
    @Last_name = 'Smith',
    @Date_of_birth = '1985-05-05',
    @NIP = 1234567890,
    @Company_name = 'TechCorp',
    @Phone = '987654321',
    @Email = 'jane.smith@techcorp.com'
```

Customer_ID	City_ID	Street	First_name	Last_name	Date_of_birth	NIP	Phone	Email
9	2	Corporate Avenue	Jane	Smith	1985-05-05	1234567890	987654321	jane.smith@techcorp.com

Procedura tworzenia rezerwacji

```
CREATE PROCEDURE InsertReservation
    @Trip_ID INT,
    @Customer_ID INT,
    @Quantity INT
AS
BEGIN
    DECLARE @Price_p_p MONEY;
    DECLARE @AvailableQuantity INT;

    -- Fetch the price per person from the Trip_offer table
    SELECT @Price_p_p = Price_p_p FROM Trip_offer WHERE Trip_ID = @Trip_ID;

    IF @Price_p_p IS NULL
    BEGIN
        RAISERROR('Trip not found', 16, 1);
        RETURN;
    END

    -- Fetch the available quantity from the Available_trips_quantity view
    SELECT @AvailableQuantity = Available_Quantity FROM Available_trips_quantity WHERE Trip_ID = @Trip_ID;

    IF @AvailableQuantity IS NULL
    BEGIN
        RAISERROR('Available quantity not found for the trip', 16, 1);
        RETURN;
    END

    -- Ensure the requested quantity does not exceed the available quantity
    IF @Quantity > @AvailableQuantity
    BEGIN
        RAISERROR('Requested quantity exceeds available spots', 16, 1);
        RETURN;
    END

    -- Insert new reservation
    INSERT INTO Reservations (Trip_ID, Customer_ID, Quantity, Price_p_p)
    VALUES (@Trip_ID, @Customer_ID, @Quantity, @Price_p_p);

    -- Get the ID of the newly inserted reservation
    DECLARE @NewReservation_ID INT = SCOPE_IDENTITY();

    PRINT 'New reservation added:';
    SELECT * FROM Reservations WHERE Reservation_ID = @NewReservation_ID;
END;
GO
```

Komentarz

Procedura pozwala dodawać nową rezerwację z wybraną wycieczką oraz liczbą miejsc. Sprawdzane jest czy liczba miejsc nie przekracza liczby dostępnych miejsc. Procedura kończy się wyświetleniem jaki wpis został dodany.

Przykładowe wywołanie procedury

```
EXEC InsertReservation
    @Trip_ID = 11,
    @Customer_ID = 7,
    @Quantity = 2;
```

Reservation_ID	Trip_ID	Customer_ID	Status	Quantity	Price_p_p
9	11	7	E	2	2500,00

Procedura edytowania rezerwacji

```
CREATE PROCEDURE UpdateReservation
    @Reservation_ID INT,
    @Trip_ID INT = NULL,
    @Customer_ID INT = NULL,
    @Quantity INT = NULL
AS
BEGIN
    DECLARE @Status CHAR(1);
    DECLARE @CurrentQuantity INT;
    DECLARE @AvailableQuantity INT;
    DECLARE @ExistingParticipants INT;

    IF @Reservation_ID IS NULL
    BEGIN
        RAISERROR('Reservation_ID is required for update', 16, 1);
        RETURN;
    END

    -- Fetch the status and current quantity from the Reservations table
    SELECT @Status = Status, @CurrentQuantity = Quantity FROM Reservations
    WHERE Reservation_ID = @Reservation_ID;

    IF @Status IS NULL
    BEGIN
        RAISERROR('Reservation not found', 16, 1);
        RETURN;
    END

    IF @Status <> 'E'
    BEGIN
        RAISERROR('Reservation can only be updated if the status is ''E''', 16, 1);
        RETURN;
    END

    -- Fetch the available quantity from the Available_trips_quantity view
    SELECT @AvailableQuantity = Available_Quantity FROM Available_trips_quantity WHERE Trip_ID = @Trip_ID;

    IF @AvailableQuantity IS NULL
    BEGIN
        RAISERROR('Available quantity not found for the trip', 16, 1);
        RETURN;
    END

    -- Check the number of existing participants for the reservation
    SELECT @ExistingParticipants = COUNT(*) FROM Participants WHERE Reservation_ID = @Reservation_ID;

    -- Ensure the new quantity does not exceed the available quantity plus the current reservation quantity
    IF @Quantity IS NOT NULL
    BEGIN
        IF @Quantity < @ExistingParticipants
        BEGIN
            RAISERROR('New quantity cannot be less than the number of existing participants', 16, 1);
            RETURN;
        END

        IF @Quantity > (@AvailableQuantity + @CurrentQuantity)
        BEGIN
            RAISERROR('Requested quantity exceeds available spots', 16, 1);
            RETURN;
        END
    END

    -- Update reservation details
    UPDATE Reservations
    SET Trip_ID = ISNULL(@Trip_ID, Trip_ID),
```

```
        Customer_ID = ISNULL(@Customer_ID, Customer_ID),
        Quantity = ISNULL(@Quantity, Quantity)
    WHERE Reservation_ID = @Reservation_ID;

    PRINT 'Reservation updated:';
    SELECT * FROM Reservations WHERE Reservation_ID = @Reservation_ID;
END;
GO
```

Komentarz

Procedura pozwala edytować rezerwację, co do ilości miejsc. Sprawdzane jest czy liczba miejsc nie przekracza liczby dostępnych miejsc oraz czy rezerwację można edytować. Procedura kończy się wyświetleniem jaki wpis został edytowany.

Przykładowe wywołanie procedury

```
EXEC UpdateReservation
    @Reservation_ID = 9,
    @Trip_ID = 11,
    @Customer_ID = 7,
    @Quantity = 3;
```

Reservation_ID	Trip_ID	Customer_ID	Status	Quantity	Price_p_p
9	11	7	E	3	2500,00

Procedura anulowania rezerwacji

```
CREATE PROCEDURE CancelReservation
    @Reservation_ID INT
AS
BEGIN
    DECLARE @Status CHAR(1);
    DECLARE @ParticipantCount INT;

    IF @Reservation_ID IS NULL
    BEGIN
        RAISERROR('Reservation_ID is required for cancellation', 16, 1);
        RETURN;
    END

    -- Fetch the status from the Reservations table
    SELECT @Status = Status FROM Reservations WHERE Reservation_ID = @Reservation_ID;

    IF @Status IS NULL
    BEGIN
        RAISERROR('Reservation not found', 16, 1);
        RETURN;
    END

    IF @Status <> 'E'
    BEGIN
        RAISERROR('Reservation can only be cancelled if the status is ''E''', 16, 1);
        RETURN;
    END

    -- Check for participants linked to the reservation
    SELECT @ParticipantCount = COUNT(*) FROM Participants WHERE Reservation_ID = @Reservation_ID;

    IF @ParticipantCount > 0
    BEGIN
        RAISERROR('Reservation cannot be cancelled as there are participants linked to it', 16, 1);
        RETURN;
    END

    -- Set the reservation status to 'C'
    UPDATE Reservations
    SET Status = 'C'
    WHERE Reservation_ID = @Reservation_ID;

    PRINT 'Reservation cancelled successfully';
END;
GO
```

Komentarz

Procedura pozwala anulować rezerwację. Aby anulować rezerwacje wystarczy, że jest ona w trybie edytowalnym. Po wykonaniu procedury zostają usunięte wszystkie informacje o członkach danej rezerwacji. Procedura kończy się wpisem na standardowe wyjście.

Przykładowe wywołanie procedury

```
EXEC CancelReservation
    @Reservation_ID = 8;
```

(1 row affected) Reservation and related data cancelled and deleted Total execution time: 00:00:00.010

Procedura przywrócenia anulowanej rezerwacji

```
CREATE PROCEDURE RestoreReservation
    @Reservation_ID INT
AS
BEGIN
    DECLARE @Status CHAR(1);
    DECLARE @Trip_ID INT;
    DECLARE @Quantity INT;
    DECLARE @Start_date DATE;

    IF @Reservation_ID IS NULL
    BEGIN
        RAISERROR('Reservation_ID is required for restoration', 16, 1);
        RETURN;
    END

    -- Fetch the reservation details
    SELECT @Status = Status, @Trip_ID = Trip_ID, @Quantity = Quantity
    FROM Reservations
    WHERE Reservation_ID = @Reservation_ID;

    IF @Status IS NULL
    BEGIN
        RAISERROR('Reservation not found', 16, 1);
        RETURN;
    END

    IF @Status <> 'C'
    BEGIN
        RAISERROR('Reservation can only be restored if the status is ''C''', 16, 1);
        RETURN;
    END

    -- Check the start date of the trip
    SELECT @Start_date = T.Start_date
    FROM Trip_offer T
    WHERE T.Trip_ID = @Trip_ID;

    IF @Start_date <= DATEADD(day, 7, GETDATE())
    BEGIN
        RAISERROR('Trip start date must be more than 7 days from today for restoration', 16, 1);
        RETURN;
    END

    -- Check if the quantity is available
    IF NOT EXISTS (
        SELECT 1
        FROM Available_trips_quantity ATQ
        WHERE ATQ.Trip_ID = @Trip_ID AND ATQ.Available_Quantity >= @Quantity
    )
    BEGIN
        RAISERROR('Insufficient available quantity for restoration', 16, 1);
        RETURN;
    END

    -- Restore the reservation status to 'E'
    UPDATE Reservations
    SET Status = 'E'
    WHERE Reservation_ID = @Reservation_ID;

    PRINT 'Reservation status restored to ''E''';
END;
GO
```

Komentarz

Procedura pozwala przywrócic anulowaną rejestracje, jeżeli wycieczka jest jeszcze możliwa do rezerwowania oraz miejsca są dostępne.

Przykładowe wywołanie procedury

```
EXEC RestoreReservation
    @Reservation_ID = 8;
```

(1 row affected) Reservation deleted Total execution time: 00:00:00.008

Procedura dodania atrakcji do danej rezerwacji

```
CREATE PROCEDURE InsertReservationDetail
    @Attraction_ID INT,
    @Reservation_ID INT,
    @Trip_ID INT,
    @Quantity INT
AS
BEGIN
    DECLARE @Price_p_p MONEY;
    DECLARE @Available_quantity INT;
    DECLARE @Reservation_Status CHAR(1);
    DECLARE @Reservation_Quantity INT;
    DECLARE @Reservation_Trip_ID INT;

    -- Check if the reservation status is 'E' and fetch related Trip_ID
    SELECT @Reservation_Status = Status,
    @Reservation_Quantity = Quantity,
    @Reservation_Trip_ID = Trip_ID
    FROM Reservations
    WHERE Reservation_ID = @Reservation_ID;

    IF @Reservation_Status IS NULL
    BEGIN
        RAISERROR('Reservation not found', 16, 1);
        RETURN;
    END

    IF @Reservation_Status <> 'E'
    BEGIN
        RAISERROR('Reservation status is not ''E''', 16, 1);
        RETURN;
    END

    -- Check if the Reservation is associated with the given Trip_ID
    IF @Reservation_Trip_ID <> @Trip_ID
    BEGIN
        RAISERROR('Reservation_ID is not associated with the given Trip_ID', 16, 1);
        RETURN;
    END

    -- Fetch the price per person from the Attractions table
    SELECT @Price_p_p = Price_p_p
    FROM Attractions
    WHERE Attraction_ID = @Attraction_ID;

    IF @Price_p_p IS NULL
    BEGIN
        RAISERROR('Attraction not found', 16, 1);
        RETURN;
    END

    -- Check if the Attraction_ID is valid for the given Trip_ID and fetch available quantity
    SELECT @Available_quantity = Quantity_Difference
    FROM Available_attractions_quantity
    WHERE Trip_ID = @Trip_ID AND Attraction_ID = @Attraction_ID;

    IF @Available_quantity IS NULL
    BEGIN
        RAISERROR('Attraction is not valid for the given Trip_ID', 16, 1);
        RETURN;
    END

    -- Check if available quantity is greater than or equal to the requested quantity
    IF @Available_quantity < @Quantity
    BEGIN
        RAISERROR('Available quantity is less than the requested quantity', 16, 1);
```

```
        RETURN;
    END

    -- Check if the requested quantity does not exceed the reservation quantity
    IF @Quantity > @Reservation_Quantity
    BEGIN
        RAISERROR('Requested quantity exceeds the reservation quantity', 16, 1);
        RETURN;
    END

    -- Insert new reservation detail
    INSERT INTO Reservations_details (Attraction_ID, Reservation_ID, Quantity, Price_p_p)
    VALUES (@Attraction_ID, @Reservation_ID, @Quantity, @Price_p_p);

    PRINT 'New reservation detail added:';
    SELECT * FROM Reservations_details
    WHERE Attraction_ID = @Attraction_ID AND Reservation_ID = @Reservation_ID;
END;
GO
```

Komentarz

Procedura pozwala dodawać atrakcje do danej rezerwacji. Sprawdzane jest czy podana ilość nie przekracza dostępnej ilości atrakcji, dodatkowo sprawdzane jest czy podana ilość nie przekracza zadeklarowanej ilości miejsc na wycieczkę.

Przykładowe wywołanie procedury

```
EXEC InsertReservationDetail
    @Attraction_ID = 26,
    @Reservation_ID = 10,
    @Trip_ID = 11,
    @Quantity = 2;
```

Attraction_ID	Reservation_ID	Quantity	Price_p_p
26	10	2	360,00

Procedura zmiany ilości danej atrakcji danej rezerwacji

```
CREATE PROCEDURE UpdateReservationDetail
    @Attraction_ID INT,
    @Reservation_ID INT,
    @Trip_ID INT,
    @Quantity INT
AS
BEGIN
    DECLARE @Price_p_p MONEY;
    DECLARE @Available_quantity INT;
    DECLARE @Reservation_Status CHAR(1);
    DECLARE @CurrentQuantity INT;
    DECLARE @ExistingAttractionParticipants INT;
    DECLARE @Reservation_Trip_ID INT;

    IF @Reservation_ID IS NULL
    BEGIN
        RAISERROR('Reservation_ID is required for update', 16, 1);
        RETURN;
    END

    -- Check if the reservation status is 'E' and fetch related Trip_ID
    SELECT @Reservation_Status = Status, @CurrentQuantity = Quantity, @Reservation_Trip_ID = Trip_ID
    FROM Reservations
    WHERE Reservation_ID = @Reservation_ID;

    IF @Reservation_Status IS NULL
    BEGIN
        RAISERROR('Reservation not found', 16, 1);
        RETURN;
    END

    IF @Reservation_Status <> 'E'
    BEGIN
        RAISERROR('Reservation status is not ''E''', 16, 1);
        RETURN;
    END

    -- Check if the Reservation is associated with the given Trip_ID
    IF @Reservation_Trip_ID <> @Trip_ID
    BEGIN
        RAISERROR('Reservation_ID is not associated with the given Trip_ID', 16, 1);
        RETURN;
    END

    -- Check if the Attraction_ID is valid for the given Trip_ID and fetch available quantity
    SELECT @Available_quantity = Quantity_Difference
    FROM Available_attractions_quantity
    WHERE Trip_ID = @Trip_ID AND Attraction_ID = @Attraction_ID;

    IF @Available_quantity IS NULL
    BEGIN
        RAISERROR('Attraction is not valid for the given Trip_ID', 16, 1);
        RETURN;
    END

    -- Check if available quantity is greater than or equal to the requested quantity
    IF @Available_quantity < @Quantity
    BEGIN
        RAISERROR('Available quantity is less than the requested quantity', 16, 1);
        RETURN;
    END

    -- Check if the requested quantity does not exceed the reservation quantity
    IF @Quantity > @CurrentQuantity
    BEGIN
```

```
        RAISERROR('Requested quantity exceeds the reservation quantity', 16, 1);
        RETURN;
    END

    -- Check the number of existing participants for the attraction associated with the reservation
    SELECT @ExistingAttractionParticipants = COUNT(*)
    FROM Attraction_for_participant afp
    JOIN Participants p ON afp.Participant_ID = p.Participant_ID
    WHERE afp.Reservation_ID = @Reservation_ID AND afp.Attraction_ID = @Attraction_ID;

    -- Ensure the new quantity does not fall below the number of existing participants
    IF @Quantity < @ExistingAttractionParticipants
    BEGIN
        RAISERROR('New quantity cannot be less than the number of existing participants', 16, 1);
        RETURN;
    END

    -- Update reservation detail without changing the price
    UPDATE Reservations_details
    SET Quantity = @Quantity
    WHERE Attraction_ID = @Attraction_ID AND Reservation_ID = @Reservation_ID;

    PRINT 'Reservation detail updated:';
    SELECT * FROM Reservations_details WHERE Attraction_ID = @Attraction_ID
    AND Reservation_ID = @Reservation_ID;
END;
GO
```

Komentarz

Procedura pozwala zmieniać ilość danej atrakcje danej rezerwacji. Sprawdzane jest czy podana ilość nie przekracza dostępnej ilości atrakcji, dodatkowo sprawdzane jest czy podana ilość nie przekracza zadeklarowanej ilości miejsc na wycieczkę.

Przykładowe wywołanie procedury

```
EXEC UpdateReservationDetail
    @Attraction_ID = 26,
    @Reservation_ID = 10,
    @Trip_ID = 11,
    @Quantity = 1;
```

Attraction_ID	Reservation_ID	Quantity	Price_p_p
26	10	1	360,00

Procedura usunięcia danej atrakcji danej rezerwacji

```
CREATE PROCEDURE DeleteReservationDetail
    @Attraction_ID INT,
    @Reservation_ID INT
AS
BEGIN
    DECLARE @Reservation_Status CHAR(1);

    IF @Reservation_ID IS NULL
    BEGIN
        RAISERROR('Reservation_ID is required for delete', 16, 1);
        RETURN;
    END

    -- Check if the reservation status is 'E'
    SELECT @Reservation_Status = Status FROM Reservations WHERE Reservation_ID = @Reservation_ID;

    IF @Reservation_Status IS NULL
    BEGIN
        RAISERROR('Reservation not found', 16, 1);
        RETURN;
    END

    IF @Reservation_Status <> 'E'
    BEGIN
        RAISERROR('Reservation status is not ''E''', 16, 1);
        RETURN;
    END

    -- Identify Participant_IDs associated with the given Attraction_ID and Reservation_ID
    DECLARE @Participant_IDs TABLE (Participant_ID INT);

    INSERT INTO @Participant_IDs (Participant_ID)
    SELECT p.Participant_ID
    FROM Participants p
    JOIN Attraction_for_participant afp ON p.Participant_ID = afp.Participant_ID
    WHERE afp.Attraction_ID = @Attraction_ID AND p.Reservation_ID = @Reservation_ID;

    -- Delete from Attractions_for_participants
    DELETE afp
    FROM Attraction_for_participant afp
    JOIN @Participant_IDs pids ON afp.Participant_ID = pids.Participant_ID
    WHERE afp.Attraction_ID = @Attraction_ID;

    -- Delete from Participants
    DELETE p
    FROM Participants p
    JOIN @Participant_IDs pids ON p.Participant_ID = pids.Participant_ID;

    -- Delete reservation detail
    DELETE FROM Reservations_details WHERE Attraction_ID = @Attraction_ID
    AND Reservation_ID = @Reservation_ID;

    PRINT 'Reservation detail and related participants deleted';
END;
GO
```

Komentarz

Procedura pozwala usuwać dane atrakcje danej rezerwacji. Spawdzane jest czy rezerwacja jest w stanie edycji.

Przykładowe wywołanie procedury

```
EXEC DeleteReservationDetail
    @Attraction_ID = 2,
    @Reservation_ID = 1;
```

(0 rows affected) (0 rows affected) (0 rows affected) (1 row affected) Reservation detail and related participants deleted Total execution time: 00:00:00.032

Procedura dodawania uczestnika do rezerwacji

```
CREATE PROCEDURE InsertParticipantAndAttraction
    @Reservation_ID INT,
    @First_name VARCHAR(30),
    @Last_name VARCHAR(30),
    @Date_of_birth DATE,
    @Attraction_ID INT = NULL
AS
BEGIN
    DECLARE @Participant_ID INT;
    DECLARE @CurrentParticipantCount INT;
    DECLARE @ReservationQuantity INT;
    DECLARE @Reservation_Status CHAR(1);
    DECLARE @CurrentAttractionParticipantCount INT;
    DECLARE @AttractionQuantity INT;
    DECLARE @TotalAttractionParticipants INT;

    -- Check the reservation status
    SELECT @Reservation_Status = Status, @ReservationQuantity = Quantity
    FROM Reservations
    WHERE Reservation_ID = @Reservation_ID;

    IF @Reservation_Status IS NULL
    BEGIN
        RAISERROR('Reservation not found', 16, 1);
        RETURN;
    END

    IF @Reservation_Status <> 'E'
    BEGIN
        RAISERROR('Reservation status is not ''E''', 16, 1);
        RETURN;
    END

    -- Check the current number of participants for the reservation
    SELECT @CurrentParticipantCount = COUNT(*)
    FROM Participants
    WHERE Reservation_ID = @Reservation_ID;

    -- Check the total number of participants for all attractions
    SELECT @TotalAttractionParticipants = ISNULL(SUM(Quantity), 0)
    FROM Reservations_details
    WHERE Reservation_ID = @Reservation_ID;

    -- Check if adding the participant exceeds the reservation quantity
    IF @CurrentParticipantCount + 1 > @ReservationQuantity
    BEGIN
        RAISERROR('Adding this participant exceeds the allowed quantity for the reservation', 16, 1);
        RETURN;
    END

    -- Check if adding the participant without an attraction exceeds the reservation quantity minus total
attraction participants
    IF @Attraction_ID IS NULL
    AND @CurrentParticipantCount + 1 > (@ReservationQuantity - @TotalAttractionParticipants)
    BEGIN
        RAISERROR('Adding this participant without an attraction exceeds the allowed quantity', 16, 1);
        RETURN;
    END

    -- Insert into Participants table
    INSERT INTO Participants (Reservation_ID, First_name, Last_name, Date_of_birth)
    VALUES (@Reservation_ID, @First_name, @Last_name, @Date_of_birth);

    -- Get the Participant_ID of the newly inserted participant
    SET @Participant_ID = SCOPE_IDENTITY();
```

```
    -- If Attraction_ID is provided, check the participants count for the attraction
    IF @Attraction_ID IS NOT NULL
    BEGIN
        -- Get the current participant count for the attraction
        SELECT @CurrentAttractionParticipantCount = COUNT(*)
        FROM Attraction_for_Participant
        WHERE Reservation_ID = @Reservation_ID AND Attraction_ID = @Attraction_ID;

        -- Get the allowed quantity for the attraction in the reservation details
        SELECT @AttractionQuantity = Quantity
        FROM Reservations_details
        WHERE Reservation_ID = @Reservation_ID AND Attraction_ID = @Attraction_ID;

        IF @AttractionQuantity IS NULL
        BEGIN
            RAISERROR('Attraction not found in reservation details', 16, 1);
            RETURN;
        END

        -- Check if adding the participant exceeds the allowed quantity for the attraction
        IF @CurrentAttractionParticipantCount + 1 > @AttractionQuantity
        BEGIN
            RAISERROR('Adding this participant exceeds the allowed quantity for the attraction', 16, 1);
            RETURN;
        END

        -- Insert into Attraction_for_Participant table
        INSERT INTO Attraction_for_Participant (Participant_ID, Reservation_ID, Attraction_ID)
        VALUES (@Participant_ID, @Reservation_ID, @Attraction_ID);

        -- Display the inserted participant and attraction details
        PRINT 'Participant and attraction for participant added successfully:';
        SELECT p.*, afp.*
        FROM Participants p
        JOIN Attraction_for_Participant afp ON p.Participant_ID = afp.Participant_ID
        WHERE p.Participant_ID = @Participant_ID AND afp.Attraction_ID = @Attraction_ID;
    END
    ELSE
    BEGIN
        -- Display the inserted participant details
        PRINT 'Participant added successfully:';
        SELECT * FROM Participants WHERE Participant_ID = @Participant_ID;
    END
END;
GO
```

Komentarz

Procedura pozwala dodawać uczestników do danej rezerwacji oraz określać ich atrakcje.

Przykładowe wywołanie procedury

```
EXEC InsertParticipantAndAttraction
    @Reservation_ID = 10,
    @First_name = 'Zofia',
    @Last_name = 'Zespolona',
    @Date_of_birth = '1967-05-15',
    @Attraction_ID = 26;
```

Participant_ID	Reservation_ID	First_name	Last_name	Date_of_birth	Participant_ID	Reservation_ID	Attraction_ID
51	10	Zofia	Zespolona	1967-05-15	51	10	26

Procedura usuwania uczestnika z rezerwacji

```
CREATE PROCEDURE DeleteParticipantAndAttraction
    @Participant_ID INT
AS
BEGIN
    DECLARE @Reservation_ID INT;
    DECLARE @Reservation_Status CHAR(1);

    IF @Participant_ID IS NULL
    BEGIN
        RAISERROR('Participant_ID is required for delete', 16, 1);
        RETURN;
    END

    -- Fetch the Reservation_ID associated with the Participant
    SELECT @Reservation_ID = Reservation_ID
    FROM Participants
    WHERE Participant_ID = @Participant_ID;

    IF @Reservation_ID IS NULL
    BEGIN
        RAISERROR('Reservation not found for the given Participant_ID', 16, 1);
        RETURN;
    END

    -- Check the reservation status
    SELECT @Reservation_Status = Status
    FROM Reservations
    WHERE Reservation_ID = @Reservation_ID;

    IF @Reservation_Status IS NULL
    BEGIN
        RAISERROR('Reservation not found', 16, 1);
        RETURN;
    END

    IF @Reservation_Status <> 'E'
    BEGIN
        RAISERROR('Reservation status is not ''E''', 16, 1);
        RETURN;
    END

    -- Delete from Attractions_for_participants
    DELETE FROM Attraction_for_participant WHERE Participant_ID = @Participant_ID;

    -- Delete from Participants
    DELETE FROM Participants WHERE Participant_ID = @Participant_ID;

    PRINT 'Participant and associated attractions deleted successfully';
END;
GO
```

Komentarz

Procedura pozwala usuwać uczestników z danej rezerwacji.

Przykładowe wywołanie procedury

```
EXEC DeleteParticipantAndAttraction
    @Participant_ID = 49;
```

(0 rows affected) (1 row affected) Participant and associated attractions deleted successfully Total execution time: 00:00:00.011

Procedura zmiany statusu rezerwacji.

```
CREATE PROCEDURE CheckAndUpdateReservations
AS
BEGIN
    DECLARE @TodayPlus7Days DATE;
    SET @TodayPlus7Days = DATEADD(day, 7, GETDATE());

    -- Update status based on Reservations_condition view
    WITH Reservations_condition_aggregated AS (
        SELECT
            Reservation_ID,
            MIN(Participants_for_trip) AS Participants_for_trip,
            MIN(Participants_for_Attraction) AS Participants_for_Attraction,
            MIN(Paid) AS Paid
        FROM Reservations_condition
        GROUP BY Reservation_ID
    )
    UPDATE Reservations
    SET Status =
        CASE
            WHEN Participants_for_trip = 'ERROR'
            OR Participants_for_Attraction = 'ERROR'
            OR Paid = 'ERROR' THEN 'C'
            ELSE 'A'
        END
    FROM Reservations R
    JOIN Reservations_condition_aggregated RCA ON R.Reservation_ID = RCA.Reservation_ID
    JOIN Trip_offer T ON R.Trip_ID = T.Trip_ID
    WHERE T.Start_date <= @TodayPlus7Days;

    PRINT 'Reservations status updated successfully';
END;
GO
```

Komentarz

Procedura zarządza statusem rezerwacji, sprawdza wszystkie rezerwacje, których wycieczka rozpoczyna się za 7 dni. Procedura może zostać przypisana do codziennego harmonogramu (JOB).

Przykładowe wywołanie procedury

```
EXEC CheckAndUpdateReservations;
```

(0 rows affected) Reservations status updated successfully Total execution time: 00:00:00.032

Procedura opłacania rezerwacji

```
CREATE PROCEDURE AddFinancialOperation
    @Reservation_ID int,
    @Amount money,
    @Type char(1)
AS
BEGIN
    -- Check if the amount is positive
    IF @Amount <= 0
    BEGIN
        RAISERROR('Amount must be positive.', 16, 1);
        RETURN;
    END

    -- Check if the operation type is valid
    IF @Type NOT IN ('C', 'M', 'T')
    BEGIN
        RAISERROR('Invalid operation type. Must be one of ''C'', ''M'', or ''T''.', 16, 1);
        RETURN;
    END

    -- Check if Reservation_ID exists in the Reservations table
    IF NOT EXISTS (SELECT 1 FROM Reservations WHERE Reservation_ID = @Reservation_ID)
    BEGIN
        RAISERROR('Reservation ID does not exist.', 16, 1);
        RETURN;
    END

    -- Add a new record to the Financial_operations table
    INSERT INTO Financial_operations (Reservation_ID, Amount, Type)
    VALUES (@Reservation_ID, @Amount, @Type);

    -- Inform about the successful operation
    PRINT 'Financial operation added successfully.';
END;
GO
```

Komentarz

Procedura pozwala dodawać opłaty za rezerwacje, określany jest typ wpłaty oraz kwota.

Przykładowe wywołanie procedury

```
EXEC AddFinancialOperation
    @Reservation_ID = 10,
    @Amount = 100.00,
    @Type = 'M';
```

Financial operation added successfully. Total execution time: 00:00:00.013

Funkcje

Funkcja wyświetlania salda danego użytkownika

```
CREATE FUNCTION GetCustomerSaldo
(
    @Customer_ID INT
)
RETURNS MONEY
AS
BEGIN
    DECLARE @Saldo MONEY;

    SELECT @Saldo = SALDO
    FROM Client_saldo
    WHERE Customer_ID = @Customer_ID;

    RETURN @Saldo;
END;
```

Komentarz

Funkcja zwraca saldo danego użytkownika.

Użycie funkcji

```
DECLARE @CustomerID INT = 3;
DECLARE @Saldo MONEY;

SET @Saldo = dbo.GetCustomerSaldo(@CustomerID);

IF @Saldo IS NOT NULL
BEGIN
    PRINT 'Saldo klienta ' + CAST(@CustomerID AS VARCHAR) + ' wynosi: ' + CAST(@Saldo AS VARCHAR);
END
ELSE
BEGIN
    PRINT 'Brak danych o saldzie dla klienta o ID: ' + CAST(@CustomerID AS VARCHAR);
END;
```

Saldo klienta 3 wynosi: -3075.00

Funkcja wyświetlania wycieczek w danym okresie czasowym

```
CREATE FUNCTION GetTripsInRange
(
    @Start_date DATE,
    @End_date DATE
)
RETURNS TABLE
AS
RETURN
(
    SELECT Trip_ID, City, Start_date, End_date
    FROM Available_trips
    WHERE Start_date BETWEEN @Start_date AND @End_date
       OR End_date BETWEEN @Start_date AND @End_date
);
```

Komentarz

Funkcja zwraca tabelę z widoku Available_trips z danego okresu.

Użycie funkcji

```
DECLARE @StartDate DATE = '2024-06-01';
DECLARE @EndDate DATE = '2024-08-15';

SELECT *
FROM GetTripsInRange(@StartDate, @EndDate);
```

Trip_ID	City	Start_date	End_date
6	Wieden	2024-07-01	2024-07-07

Funkcja wyświetlania uczestników wybranego klienta z danej wycieczki

```
CREATE FUNCTION GetParticipantsForCustomer
(
    @Customer_ID INT
)
RETURNS TABLE
AS
RETURN
(
    SELECT
        P.Participant_ID,
        P.Reservation_ID,
        P.First_name,
        P.Last_name,
        P.Date_of_birth,
        A.Attraction_ID
    FROM
        Customers C
    JOIN
        Reservations R ON C.Customer_ID = R.Customer_ID
    JOIN
        Participants P ON R.Reservation_ID = P.Reservation_ID
    LEFT JOIN
        Attraction_for_participant A ON P.Participant_ID = A.Participant_ID
    WHERE
        C.Customer_ID = @Customer_ID
);
```

Komentarz

Funkcja zwraca tabelę z informacją o rezerwacjach i ich uczestnikach dla danego klienta.

Użycie funkcji

```
DECLARE @CustomerID INT = 7;

SELECT *
FROM GetParticipantsForCustomer(@CustomerID);
```

Participant_ID	Reservation_ID	First_name	Last_name	Date_of_birth	Attraction_ID
18	5	Jolanta	Rola	1980-01-01	16
19	5	Adam	Rola	1981-01-01	16
20	5	Fiona	Serce	1982-01-01	17
21	5	Samanta	Pudelko	1983-01-01	17

Triggery

Triger sprawdzający czy atrakcje powiązane z wycieczką zawierają się w czasie wycieczki

```
CREATE TRIGGER trgCheckAttractionDates
ON Attractions
FOR INSERT, UPDATE
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @count INT;

    -- Check if any inserted or updated attractions have dates outside the trip dates
    SELECT @count = COUNT(*)
    FROM inserted i
    JOIN Trip_offer t ON i.Trip_ID = t.Trip_ID
    WHERE i.Start_date < t.Start_date
       OR i.End_date > t.End_date;

    -- If count is greater than zero, raise an error and rollback the transaction
    IF @count > 0
    BEGIN
        RAISERROR('Attraction dates must be within the trip dates.', 16, 1);
        ROLLBACK TRANSACTION;
    END
END;
GO
```

Komentarz

Triger zapewnia spójność danych, aby dana atrakcja przypisana do danej wycieczki nie mogła być z poza zakresu czasu wycieczki.

Triger sprawdzający czy atrakcje powiązane z wycieczką nie przekraczają ilości miejsc na wycieczkę

```
CREATE TRIGGER trgCheckAttractionQuantity
ON Attractions
FOR INSERT, UPDATE
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @count INT;

    -- Check if any inserted or updated attractions have quantity greater than the trip quantity
    SELECT @count = COUNT(*)
    FROM inserted i
    JOIN Trip_offer t ON i.Trip_ID = t.Trip_ID
    WHERE i.Quantity > t.Quantity;

    -- If count is greater than zero, raise an error and rollback the transaction
    IF @count > 0
    BEGIN
        RAISERROR('Attraction quantity cannot exceed trip quantity.', 16, 1);
        ROLLBACK TRANSACTION;
    END
END;
GO
```

Komentarz

Triger zapewnia spójność danych, aby dana atrakcja przypisana do danej wycieczki nie mogła być z poza zakresu czasu wycieczki.

4. Inne

Uprawnienia

Definiowane role

- Administrator
- Pracownik
- Klient
- Agent_systemowy

```
CREATE ROLE [Administrator]
GO

ALTER ROLE [db_owner] ADD MEMBER [Administrator]
GO

CREATE ROLE [Pracownik]
GO

CREATE ROLE [Klient]
GO

CREATE ROLE [Agent_systemowy]
GO
```

Uprawnienia do tabel (wszystkich)

Rola	Poziom uprawnień
Administrator	właściciel
Pracownik	zmiana/odczyt
Klient	brak
Agent_systemowy	brak

Przykład nadawania uprawnień

```
GRANT SELECT, INSERT, UPDATE, DELETE ON Attraction_for_participant TO Pracownik;
GRANT SELECT, INSERT, UPDATE, DELETE ON Attractions TO Pracownik;
```

Uprawnienia do widoków

- All_accepted_participants,
- Client_reservation_saldo,
- Client_saldo,
- Client_saldo_hist,
- Customer_accepted_reservation,
- Financial_report,
- Participants_By_Age,
- Reservations_condition

Rola	Poziom uprawnień
Administrator	właściciel
Pracownik	zmiana/odczyt
Klient	brak
Agent_systemowy	brak

Przykład nadawania uprawnień

```
DENY SELECT, INSERT, UPDATE, DELETE ON All_accepted_participants TO Klient;
DENY SELECT, INSERT, UPDATE, DELETE ON Client_reservation_saldo TO Klient;
```

- Available_attractions,
- Available_attractions_quantity,
- Available_trips,
- Available_trips_quantity

Rola	Poziom uprawnień
Administrator	właściciel
Pracownik	zmiana/odczyt
Klient	odczyt
Agent_systemowy	brak

Przykład nadawania uprawnień

```
GRANT ALL ON Available_attractions_quantity TO Administrator;
GRANT SELECT, UPDATE ON Available_attractions_quantity TO Pracownik;
```

Uprawnienia do procedur/funkcji

- CheckAndUpdateReservations

Rola	Poziom uprawnień
Administrator	właściciel
Pracownik	brak
Klient	brak
Agent_systemowy	wykonanie

Przykład nadawania uprawnień

```
DENY EXECUTE ON CheckAndUpdateReservations TO Klient;
GRANT EXECUTE ON CheckAndUpdateReservations TO Agent_systemowy;
```

- DeleteParticipantAndAttraction
- DeleteReservation
- DeleteReservationDetail
- InsertParticipantAndAttraction
- InsertReservation
- InsertReservationDetail
- UpdateReservation
- UpdateReservationDetail
- AddFinancialOperation
- GetParticipantsForCustomer
- GetTripsInRange
- GetCustomerSaldo

Rola	Poziom uprawnień
Administrator	właściciel
Pracownik	wykonanie
Klient	wykonanie
Agent_systemowy	brak

Przykład nadawania uprawnień

```
GRANT EXECUTE ON InsertReservationDetail TO Klient;
DENY EXECUTE ON InsertReservationDetail TO Agent_systemowy;
GRANT SELECT ON GetParticipantsForCustomer TO Klient;
```

- InsertAttraction
- UpdateAttraction
- InsertTripOffer
- UpdateTripOffer

Rola	Poziom uprawnień
Administrator	właściciel
Pracownik	wykonanie
Klient	brak
Agent_systemowy	brak

```
GRANT EXECUTE ON InsertTripOffer TO Administrator;
GRANT EXECUTE ON InsertTripOffer TO Pracownik;
```