Baza danych firmy świadczącej usługi turystyczne (sprzedaż wycieczek) dla klientów indywidualnych oraz firm

1. Wymagania i funkcje systemu

Lista wymagań systemu:

- System musi umożliwić dokonanie klientowi (zarówno osobom indywidualnym jak i firmom) rezerwacji wycieczki.
- Klienci mogą dokonać rezerwacji dla wielu osób (uczestników wycieczki).
- Klienci mogą wybrać dodatkowe atrackje przypisane do danej oferty.
- Nie można dokonać rezerwacji dodatkowej usługi bez wcześniejszej rezerwacji wycieczki.
- W momencie dokonywania rezerwacji przez klienta może podać on tylko liczbę uczestników wycieczki oraz dodatkowo zamówione usługi.
- Tydzień przed wyjazdem klient musi podać dane identyfikujące uczestników wycieczki oraz atrakcji (w przypadku braku- anulowanie zamówienia).
- Klient może dokonywać zmian dotyczących jego zamówienia do 7 dni przed wyajzdem.
- Klient musi opłacić zamówienie do 7 dni przed terminem wyjazdu (w przypadku braku- anulowanie zamówienia).
- System zapisuje płatności powiązane z rezerwacjami (kto, kiedy, ile zapłacił i za co).

Uprawnienia użytkowników systemu:

1. Administrator systemu: - dowolna modyfikacja bazy danych

2. Klient: - przeglądać dostępne oferty wycieczek, - dokonywać rezerwacji wycieczek, - wybierać dodatkowe atrakcje przypisane do wycieczki, - dodawać uczestników do rezerwacji, - dokonywać płatności, - odwołać rezerwację wycieczki lub poszczególnej aktywności - przeglądać swoje rezerwacje i ich status, - modyfikować dane rezerwacji (do 7 dni przed wyjazdem).

3. Pracownik: - dodawać i modyfikować oferty wycieczek, - przypisywać atrakcje do ofert, - przypisywać hotele, agencje i środki transportu do ofert, - przeglądać wszystkie rezerwacje i uczestników, - generować raporty (statystyki rezerwacji, płatności, popularność ofert), - anulować lub przywracać rezerwacje ręcznie.

User Stories:

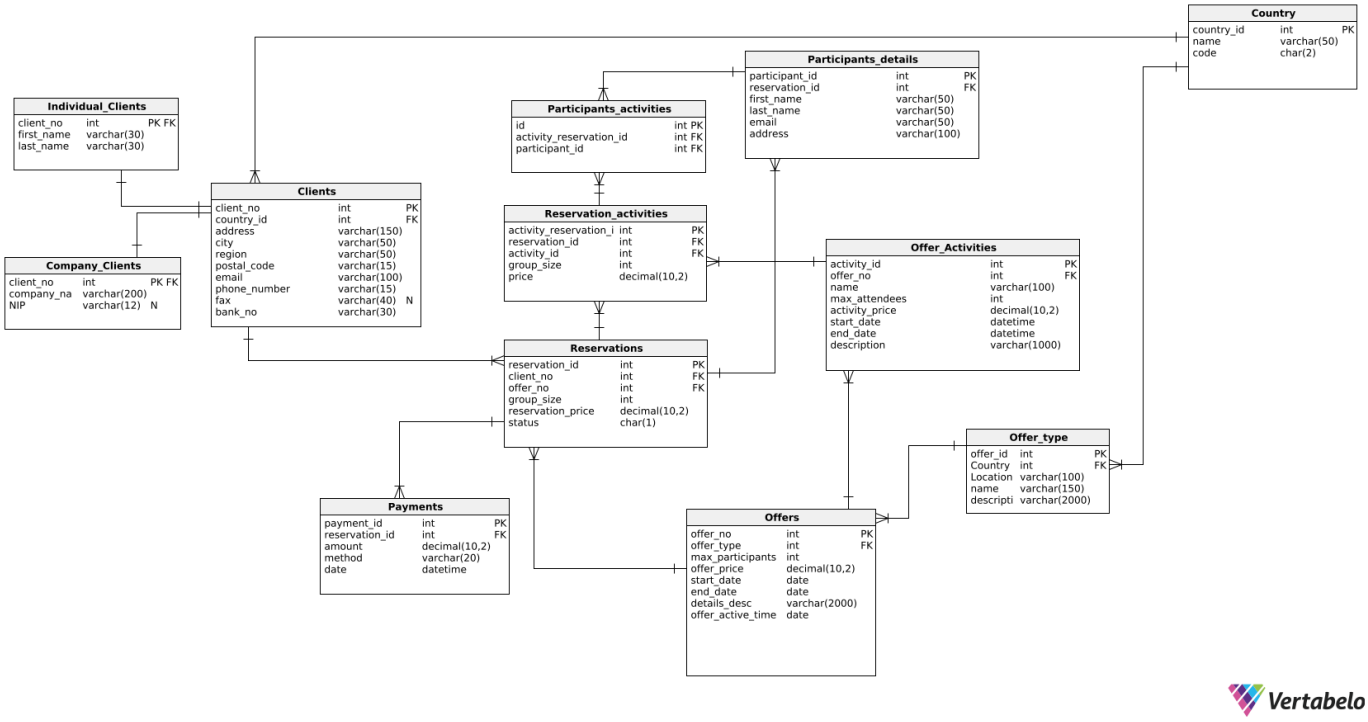
- Jako administrator, chcę eksportować dane z bazy do CSV w celu generowania raportów
- Jako pracownik chcę mieć, możliwość dodawania nowych ofert wycieczek oraz nowych dodatkowych aktywności w celu aktualizacji oferty firmy.
- Jako użytkownik chcę zarezerwować wycieczkę dla grupy osób oraz móc wybrać dodatkowe aktywności.
- Jako użytkownik chcę opłacić wycieczkę w ratach, w celu lepszego zarządzania swoim budżetem.
- Jako użytkownik chcę filtrować oraz sortować wycieczki pod względem ich bazowej ceny, miejsca docelowego, środku transportu, ocenach użytkowników, a także pod względem dostępności interesujących mnie aktywności dodatkowych, w celu łatwiejszego znalezienia najlepszej oferty.

User Stories v2

- rejestracja rezerwacji przez klienta z podaniem liczby uczestników
- dodanie danych użytkowników wycieczki
- wybór dodatkowych atrakcji przez klienta
- dokonanie płatnosci przez klienta
- automatyczne anulowanie rezerwacji przez system
- anulowanie rezerwacji przez klienta
- dodanie oferty przez adminstratora

2. Baza Danych

Schemat bazy danych:



Opis poszczególnych tabel:

Clients

Tabela zawierająca dane kontaktowe i adresowe wszystkich klientów, zarówno indywidualnych, jak i firmowych.

Nazwa Atrybutu	Typ	Opis/Uwagi
client_no	int	Klucz główny, identyfikator klienta
country_id	int	Klucz obcy do tabeli Country
address	varchar(150)	Adres klienta
city	varchar(50)	Miasto
region	varchar(50)	Region lub województwo
postal_code	varchar(15)	Kod pocztowy
email	varchar(100)	Adres email
phone_number	varchar(15)	Numer telefonu kontaktowego
fax	varchar(40)	Numer faksu (opcjonalnie)
bank_no	varchar(50)	Numer konta bankowego

- Kod DDL:

```
-- Table: Clients
CREATE TABLE Clients (
    client_no int  NOT NULL IDENTITY(1, 1),
    country_id int  NOT NULL,
    address varchar(150)  NOT NULL,
    city varchar(50)  NOT NULL,
    region varchar(50)  NOT NULL,
    postal_code varchar(15)  NOT NULL CHECK (
    PATINDEX('%[^A-Za-z0-9-]%', postal_code) = 0
    AND LEN(postal_code) BETWEEN 5 AND 15),
    email varchar(100)  NOT NULL CHECK (email LIKE '%@%.%'),
    phone_number varchar(15)  NOT NULL CHECK (LEN(phone_number) BETWEEN 7 AND 15
```

```
    AND PATINDEX('%[^0-9()+ -]%', phone_number) = 0),
    fax varchar(40)  NULL,
    bank_no varchar(50)  NOT NULL CHECK (LEN(bank_no) BETWEEN 1 AND 50 AND PATINDEX('%[^A-Z0-9 -]%', bank_no) = 0),
    CONSTRAINT client_no PRIMARY KEY  (client_no)
);
```

```
-- Reference: Country_Clients (table: Clients)
ALTER TABLE Clients ADD CONSTRAINT Country_Clients
    FOREIGN KEY (country_id)
    REFERENCES Country (country_id);
```

Company_Clients

Tabela przechowująca dane klientów firmowych, będąca specjalizacją tabeli Clients.

Nazwa Atrybutu	Typ	Opis/Uwagi
client_no	int	Klucz główny, klucz obcy do Clients
company_name	varchar(200)	Nazwa firmy
NIP	varchar(12)	Numer NIP (opcjonalnie)

- Kod DDL:

```
-- Table: Company_Clients
CREATE TABLE Company_Clients (
    client_no int  NOT NULL,
    company_name varchar(200)  NOT NULL CHECK (company_name <> ''),
    NIP varchar(12)  NULL CHECK (NIP IS NULL OR LEN(NIP) = 10 OR LEN(NIP) = 12),
    CONSTRAINT Company_Clients_pk PRIMARY KEY  (client_no)
);
```

```
-- Reference: Company_Clients_Clients (table: Company_Clients)
ALTER TABLE Company_Clients ADD CONSTRAINT Company_Clients_Clients
    FOREIGN KEY (client_no)
    REFERENCES Clients (client_no);
```

Individual_Clients

Tabela przechowująca dane osobowe klientów indywidualnych, będąca specjalizacją tabeli Clients.

Nazwa Atrybutu	Typ	Opis/Uwagi
client_no	int	Klucz główny, klucz obcy do Clients
first_name	varchar(30)	Imię klienta
last_name	varchar(30)	Nazwisko klienta

- kod DDL

```
-- Table: Individual_Clients
CREATE TABLE Individual_Clients (
    client_no int  NOT NULL,
    first_name varchar(30)  NOT NULL,
    last_name varchar(30)  NOT NULL,
    CONSTRAINT Individual_Clients_pk PRIMARY KEY  (client_no)
);
```

```
-- Reference: Individual_Clients_Clients (table: Individual_Clients)
ALTER TABLE Individual_Clients ADD CONSTRAINT Individual_Clients_Clients
    FOREIGN KEY (client_no)
    REFERENCES Clients (client_no);
```

Country

Lista krajów dostępnych w systemie, do których przypisani są klienci i oferty.

Nazwa Atrybutu	Typ	Opis/Uwagi
country_id	int	Klucz główny
name	varchar(50)	Nazwa kraju
code	char(2)	Dwuliterowy kod kraju

- kod DDL

```
-- Table: Country
CREATE TABLE Country (
    country_id int  NOT NULL IDENTITY(1, 1),
    name varchar(50)  NOT NULL,
    code char(2)  NOT NULL CHECK (code LIKE '[A-Z][A-Z]' and (LEN(code) = 2)),
    CONSTRAINT Country_pk PRIMARY KEY  (country_id)
);
```

Reservations

Tabela przechowująca informacje o dokonanych rezerwacjach przez klientów.

Nazwa Atrybutu	Typ	Opis/Uwagi
reservation_id	int	Klucz główny
client_no	int	Klucz obcy do Clients
offer_no	int	Klucz obcy do Offers
group_size	int	Liczba osób w grupie
reservation_price	decimal(10,2)	Cena całkowita rezerwacji
status	char(1)	Status rezerwacji: A - anulowane przez pracownika, K - anulowane przez klienta, D - do zapłaty , Z - zapłacono, N - upłynął termin zapłaty

- kod DDL

```
-- Table: Reservations
CREATE TABLE Reservations (
    reservation_id int  NOT NULL IDENTITY(1, 1),
    client_no int  NOT NULL,
    offer_no int  NOT NULL,
    group_size int  NOT NULL CHECK (group_size > 0),
    reservation_price decimal(10,2)  NOT NULL CHECK (reservation_price >= 0),
    status char(1)  NOT NULL CHECK (status in ('A', 'K', 'Z', 'D', 'N')),
    CONSTRAINT Reservations_pk PRIMARY KEY  (reservation_id)
);
```

```
-- Reference: Clients_Reservations (table: Reservations)
ALTER TABLE Reservations ADD CONSTRAINT Clients_Reservations
    FOREIGN KEY (client_no)
    REFERENCES Clients (client_no);
```

```
-- Reference: Offers_Reservations (table: Reservations)
ALTER TABLE Reservations ADD CONSTRAINT Offers_Reservations
    FOREIGN KEY (offer_no)
    REFERENCES Offers (offer_no);
```

Payments

Tabela przechowująca informacje o płatnościach za rezerwacje.

Nazwa Atrybutu	Typ	Opis/Uwagi
payment_id	int	Klucz główny
reservation_id	int	Klucz obcy do Reservations
amount	decimal(10,2)	Kwota płatności
method	varchar(20)	Metoda płatności (np. karta, przelew)
date	datetime	Data dokonania płatności

- kod DDL

```
-- Table: Payments
CREATE TABLE Payments (
    payment_id int  NOT NULL IDENTITY(1, 1),
    reservation_id int  NOT NULL,
    amount decimal(10,2)  NOT NULL CHECK (amount >= 0),
    method varchar(20)  NOT NULL CHECK (method IN ('BLIK', 'Karta kredytowa', 'PayPal', 'Gotówka', 'Przelew')),
    date datetime  NOT NULL,
    CONSTRAINT Payments_pk PRIMARY KEY  (payment_id)
);
```

```
-- Reference: Payments_Reservations (table: Payments)
ALTER TABLE Payments ADD CONSTRAINT Payments_Reservations
    FOREIGN KEY (reservation_id)
    REFERENCES Reservations (reservation_id);
```

Offers

Tabela przechowująca główne oferty dostępne dla klientów.

Nazwa Atrybutu	Typ	Opis/Uwagi
offer_no	int	Klucz główny
offer_type	int	Klucz obcy do Offer_type
max_participants	int	Maksymalna liczba uczestników
offer_price	decimal(10,2)	Cena podstawowa oferty
start_date	date	Data rozpoczęcia
end_date	date	Data zakończenia
details_desc	varchar(2000)	Szczegóły oferty
offer_active_time	date	Data aktywacji oferty

- kod DDL

```
-- Table: Offers
CREATE TABLE Offers (
    offer_no int  NOT NULL IDENTITY(1, 1),
    offer_type int  NOT NULL,
    max_participants int  NOT NULL CHECK (max_participants > 0),
    offer_price decimal(10,2)  NOT NULL CHECK (offer_price >= 0 ),
    start_date date  NOT NULL,
    end_date date  NOT NULL,
    CONSTRAINT check_date1 CHECK (start_date <= end_date),
    details_desc varchar(2000)  NOT NULL,
    offer_active_time date  NOT NULL,
    CONSTRAINT Offers_pk PRIMARY KEY  (offer_no)
);
```

```
-- Reference: Offers_Offer_type (table: Offers)
ALTER TABLE Offers ADD CONSTRAINT Offers_Offer_type
```

```
    FOREIGN KEY (offer_type)
    REFERENCES Offer_type (offer_id);
```

Offer_type

Tabela zawierająca informacje o typach dostępnych ofert.

Nazwa Atrybutu	Typ	Opis/Uwagi
offer_id	int	Klucz główny
Country	int	Klucz obcy do Country
Location	varchar(100)	Lokalizacja oferty
name	varchar(150)	Nazwa typu oferty
description	varchar(2000)	Opis typu oferty

- kod DDL

```
-- Table: Offer_type
CREATE TABLE Offer_type (
    offer_id int  NOT NULL IDENTITY(1, 1),
    Country int  NOT NULL,
    Location varchar(100)  NOT NULL,
    name varchar(150)  NOT NULL,
    description varchar(2000)  NOT NULL,
    CONSTRAINT offer_id PRIMARY KEY  (offer_id)
);
```

```
-- Reference: Country_Offer_type (table: Offer_type)
ALTER TABLE Offer_type ADD CONSTRAINT Country_Offer_type
    FOREIGN KEY (Country)
    REFERENCES Country (country_id);
```

Offer_Activities

Tabela przechowująca informacje o aktywnościach dostępnych w ramach oferowanych pakietów.

Nazwa Atrybutu	Typ	Opis/Uwagi
activity_id	int	Klucz główny
offer_no	int	Klucz obcy do Offers
name	varchar(100)	Nazwa aktywności
max_attendees	int	Maksymalna liczba uczestników
activity_price	decimal(10,2)	Cena udziału w aktywności
start_date	datetime	Data rozpoczęcia aktywności
end_date	datetime	Data zakończenia aktywności
description	varchar(1000)	Opis aktywności

- kod DDL

```
-- Table: Offer_Activities
CREATE TABLE Offer_Activities (
    activity_id int  NOT NULL IDENTITY(1, 1),
    offer_no int  NOT NULL,
    name varchar(100)  NOT NULL,
    max_attendees int  NOT NULL CHECK (max_attendees >= 0),
    activity_price decimal(10,2)  NOT NULL CHECK (activity_price >= 0),
    start_date datetime  NOT NULL,
    end_date datetime  NOT NULL,
    CONSTRAINT check_date2 CHECK (start_date <= end_date),
    description varchar(1000)  NOT NULL,
```

```
    CONSTRAINT Offer_Activities_pk PRIMARY KEY  (activity_id)
);
```

```
-- Reference: Offers_Offer_Activities (table: Offer_Activities)
ALTER TABLE Offer_Activities ADD CONSTRAINT Offers_Offer_Activities
    FOREIGN KEY (offer_no)
    REFERENCES Offers (offer_no);
```

Reservations_activities

Rejestruje aktywności przypisane do rezerwacji.

Nazwa Atrybutu	Typ	Opis
activity_reservation_id	int	Klucz główny
reservation_id	int	Klucz obcy do Reservations
activity_id	int	Klucz obcy do Offer_Activities
group_size	int	Liczba uczestników danej aktywności
price	decimal(10,2)	Cena aktywności

- kod DDL

```
-- Table: Reservation_activities
CREATE TABLE Reservation_activities (
    activity_reservation_id int  NOT NULL IDENTITY(1, 1),
    reservation_id int  NOT NULL,
    activity_id int  NOT NULL,
    group_size int  NOT NULL CHECK (group_size > 0),
    price decimal(10,2)  NOT NULL CHECK (price >= 0),
    CONSTRAINT id PRIMARY KEY  (activity_reservation_id)
);
```

```
-- Reference: Reservation_activities_Offer_Activities (table: Reservation_activities)
ALTER TABLE Reservation_activities ADD CONSTRAINT Reservation_activities_Offer_Activities
    FOREIGN KEY (activity_id)
    REFERENCES Offer_Activities (activity_id);
```

```
-- Reference: Reservations_Reservation_activities (table: Reservation_activities)
ALTER TABLE Reservation_activities ADD CONSTRAINT Reservations_Reservation_activities
    FOREIGN KEY (reservation_id)
    REFERENCES Reservations (reservation_id);
```

Participants_activities

Tabela zawierająca powiązania między uczestnikami a przypisanymi im aktywnościami.

Nazwa Atrybutu	Typ	Opis
id	int	Klucz główny
activity_reservation_id	int	Klucz obcy do Reservation_activities
participant_id	int	Klucz obcy do Participants_details

- kod DDL

```
-- Table: Participants_activities
CREATE TABLE Participants_activities (
    id int  NOT NULL IDENTITY(1, 1),
    activity_reservation_id int  NOT NULL,
    participant_id int  NOT NULL,
    CONSTRAINT Participants_activities_pk PRIMARY KEY  (id)
```

```
);
```

```
-- Reference: Reservation_activities_Participants_activities (table: Participants_activities)
ALTER TABLE Participants_activities ADD CONSTRAINT Reservation_activities_Participants_activities
    FOREIGN KEY (activity_reservation_id)
    REFERENCES Reservation_activities (activity_reservation_id);
```

```
-- Reference: Participants_activities_Participants_details (table: Participants_activities)
ALTER TABLE Participants_activities ADD CONSTRAINT Participants_activities_Participants_details
    FOREIGN KEY (participant_id)
    REFERENCES Participants_details (participant_id);
```

Participants_details

Tabela zawierająca szczegółowe dane uczestników.

Nazwa Atrybutu	Typ	Opis
participant_id	int	Klucz główny
reservation_id	int	Klucz obcy do Reservations
first_name	varchar(50)	Imię uczestnika
last_name	varchar(50)	Nazwisko uczestnika
email	varchar(50)	E-mail uczestnika
address	varchar(100)	Adres uczestnika

- kod DDL

```
-- Table: Participants_details
CREATE TABLE Participants_details (
    participant_id int  NOT NULL IDENTITY(1, 1),
    reservation_id int  NOT NULL,
    first_name varchar(50) NOT NULL CHECK (LEN(first_name) >= 2 OR first_name IS NULL),
    last_name varchar(50) NOT NULL CHECK (LEN(last_name) >= 2 OR last_name IS NULL),
    email varchar(50) NOT NULL CHECK (email LIKE '%@%.%'),
    address varchar(100) NOT NULL,
    CONSTRAINT Participants_details_pk PRIMARY KEY  (participant_id)
);
```

```
-- Reference: Participants_details_Reservations (table: Participants_details)
ALTER TABLE Participants_details ADD CONSTRAINT Participants_details_Reservations
    FOREIGN KEY (reservation_id)
    REFERENCES Reservations (reservation_id);
```

---

3. Widoki, procedury/funkcje, triggery

Widoki

Raport stanu aktywności – liczba rezerwacji, uczestników, wpływy.

```
CREATE VIEW v_Activity_Status_Report AS
WITH Activity_Price_Calc AS (
    SELECT
        ra.activity_id,
        SUM(ra.group_size * ra.price) AS total_activity_income,
        COUNT(DISTINCT ra.reservation_id) AS total_reservations,
        SUM(ra.group_size) AS total_participants
    FROM Reservation_activities ra
```

```
    GROUP BY ra.activity_id
)
SELECT
    oa.activity_id,
    oa.name AS activity_name,
    oa.offer_no,
    oa.max_attendees,
    COALESCE(apc.total_reservations, 0) AS total_reservations,
    COALESCE(apc.total_participants, 0) AS total_participants,
    COALESCE(apc.total_activity_income, 0) AS total_income_from_activity
FROM Offer_Activities oa
JOIN Activity_Price_Calc apc ON oa.activity_id = apc.activity_id;
```

Widok dostępnych ofert z ilością wolnych miejsc oraz ceną oferty.

```
CREATE VIEW v_Available_Offers AS
SELECT
    o.offer_no,
    (SELECT ot.name
     FROM Offer_type ot
     WHERE ot.offer_id = o.offer_type) AS offer_type_name,
    o.start_date,
    o.end_date,
    o.offer_price,
    o.max_participants,
    (
        SELECT COALESCE(SUM(r.group_size), 0)
        FROM Reservations r
        WHERE r.offer_no = o.offer_no
    ) AS reserved_places,
    o.max_participants - (
        SELECT COALESCE(SUM(r.group_size), 0)
        FROM Reservations r
        WHERE r.offer_no = o.offer_no
    ) AS available_places
FROM Offers o
WHERE GETDATE() + 7 < start_date;
```

Widok pokazujące numery rezerwacji oraz ofert i dane klienta, których rezerwacje zostały anulowane poprze brak uiszczenia opłaty do 7 dni przed startem oferty lub nieuzupełnieniem danych uczestników wycieczki do 7 dni przed startem wycieczki.

```
CREATE VIEW v_Cancelled_Auto_Clients_Reservation AS
SELECT
    c.client_no,
    ic.first_name,
    ic.last_name,
    cc.company_name,
    r.reservation_id,
    r.offer_no,
    r.status,
    o.start_date
FROM Clients c
LEFT JOIN Individual_Clients ic ON c.client_no = ic.client_no
LEFT JOIN Company_Clients cc ON c.client_no = cc.client_no
JOIN Reservations r ON c.client_no = r.client_no
JOIN Offers o ON r.offer_no = o.offer_no
WHERE
    r.status = 'N'
    AND DATEADD(DAY, -7, o.start_date) <= GETDATE()
    AND (
        NOT EXISTS (
            SELECT 1
            FROM Payments p
            WHERE p.reservation_id = r.reservation_id
        )
        OR
        NOT EXISTS (
            SELECT 1
            FROM Participants_details pd
            WHERE pd.reservation_id = r.reservation_id
```

```
        )
        );
```

Pełna informacja o zamówieniu klienta wraz z danymi o wycieczce i aktywnościami oraz ich uczestnikami (kto kupił, kto uczestniczył, jaka wycieczka, jakie aktywności, daty oraz ceny jednostkowe).

```
CREATE VIEW v_Client_and_Participants_Full_Reservation_Info AS
SELECT  c.client_no,
        ic.first_name,
        ic.last_name,
        cc.company_name,
        r.reservation_id,
        r.offer_no,
        ot.name,
        ot.description,
        o.start_date,
        o.end_date,
        o.offer_price,
        p.participant_id,
        p.first_name   AS participant_first_name,
        p.last_name    AS participant_last_name,
        p.email        AS participant_email,
        p.address      AS participant_address,
        oa.activity_id,
        oa.name        AS activity_name,
        oa.description AS activity_description,
        oa.start_date  AS activity_start_date,
        oa.end_date    AS activity_end_date,
        oa.activity_price

FROM Reservations r
JOIN Clients c ON r.client_no = c.client_no
LEFT JOIN Individual_Clients ic ON c.client_no = ic.client_no
LEFT JOIN Company_Clients cc ON c.client_no = cc.client_no
JOIN Offers o on r.offer_no = o.offer_no
JOIN Offer_type ot on o.offer_type = ot.offer_id
JOIN Participants_details p ON p.reservation_id = r.reservation_id
JOIN Participants_activities pa ON pa.participant_id = p.participant_id
JOIN Offer_Activities oa ON oa.activity_id = pa.activity_reservation_id
JOIN Reservation_activities ra ON ra.activity_reservation_id = pa.activity_reservation_id;
```

Informacja pod jaką rezerwacje przypisane są osoby biorące udział w aktywnościach wraz z ich danymi oraz nazwami aktywności.

```
CREATE VIEW v_Client_and_Participants_info AS
SELECT
    r.reservation_id,
    c.client_no,
    ic.fi
    pd.participant_id,
    pd.first_name AS participant_first_name,
    pd.last_name AS participant_last_name,
    pd.email AS participant_email,
    pd.address AS participant_address,
    oa.name
FROM Reservations r
JOIN Clients c ON r.client_no = c.client_no
LEFT JOIN Company_Clients cc ON c.client_no = cc.client_no
LEFT JOIN Individual_Clients ic ON c.client_no = ic.client_no
JOIN Participants_details pd ON r.reservation_id = pd.reservation_id
JOIN Reservation_activities ra on r.reservation_id = ra.reservation_id
JOIN Offer_Activities oa ON ra.activity_id = oa.activity_id;
```

Informacja o całkowitym balansie klienta (widok dla księgowego).

```
CREATE VIEW v_Client_Balance AS
SELECT
        c.client_no,
```

```
        ic.first_name,
        ic.last_name,
        cc.company_name,
        (r.group_size * r.reservation_price) AS reservation_price_total,
        (
        SELECT COALESCE(SUM(ra.group_size * ra.price), 0)
        FROM Reservation_activities ra
        WHERE ra.reservation_id = r.reservation_id
        ) AS activities_price_total,
    (
        (r.group_size * r.reservation_price) +
        (
            SELECT COALESCE(SUM(ra.group_size * ra.price), 0)
            FROM Reservation_activities ra
            WHERE ra.reservation_id = r.reservation_id
        )
    ) AS total_price,
        (
        SELECT COALESCE(SUM(p.amount), 0)
        FROM Payments p
        WHERE p.reservation_id = r.reservation_id
        ) AS amount_paid,
    (
        (r.group_size * r.reservation_price) +
        (
            SELECT COALESCE(SUM(ra.group_size * ra.price), 0)
            FROM Reservation_activities ra
            WHERE ra.reservation_id = r.reservation_id
        )
        -
        (
            SELECT COALESCE(SUM(p.amount), 0)
            FROM Payments p
            WHERE p.reservation_id = r.reservation_id
        )
    ) AS amount_due
FROM Reservations r
JOIN Clients C ON r.client_no = C.client_no
LEFT JOIN Individual_Clients ic ON c.client_no = ic.client_no
LEFT JOIN Company_Clients cc ON c.client_no = cc.client_no;
```

Informacja o balansie klienta (widok dla księgowego) dla danej rezerwacji.

```
CREATE VIEW v_Client_Balance_on_reservations AS
SELECT
    r.reservation_id,
    c.client_no,
    ic.first_name,
    ic.last_name,
    cc.company_name,
    r.reservation_id,
    (r.group_size * r.reservation_price) AS reservation_price_total,
    (
        SELECT COALESCE(SUM(ra.group_size * ra.price), 0)
        FROM Reservation_activities ra
        WHERE ra.reservation_id = r.reservation_id
    ) AS activities_price_total,
    (
        (r.group_size * r.reservation_price) +
        (
            SELECT COALESCE(SUM(ra.group_size * ra.price), 0)
            FROM Reservation_activities ra
            WHERE ra.reservation_id = r.reservation_id
        )
    ) AS total_price,
    (
        SELECT COALESCE(SUM(p.amount), 0)
        FROM Payments p
        WHERE p.reservation_id = r.reservation_id
    ) AS amount_paid,
    (
        (r.group_size * r.reservation_price) +
        (
            SELECT COALESCE(SUM(ra.group_size * ra.price), 0)
```

```
            FROM Reservation_activities ra
            WHERE ra.reservation_id = r.reservation_id
        ) -
        (
            SELECT COALESCE(SUM(p.amount), 0)
            FROM Payments p
            WHERE p.reservation_id = r.reservation_id
        )
    ) AS amount_due
FROM Reservations r
JOIN Clients c ON r.client_no = c.client_no
LEFT JOIN Individual_Clients ic ON c.client_no = ic.client_no
LEFT JOIN Company_Clients cc ON c.client_no = cc.client_no;
```

Lista wszystkich rezerwacji z danymi klienta oraz ceną całkowitą.

```
CREATE VIEW v_Client_Reservations AS
SELECT
    r.client_no,
    ic.first_name,
    ic.last_name,
    cc.company_name,
    c.email,
    c.phone_number,
    r.reservation_id,
    r.offer_no,
    r.group_size,
    (
        (r.group_size * r.reservation_price) +
        (
            SELECT COALESCE(SUM(ra.group_size * ra.price), 0)
            FROM Reservation_activities ra
            WHERE ra.reservation_id = r.reservation_id
        )
    ) AS total_price,
    r.status
FROM Reservations r
JOIN Clients c ON r.client_no = c.client_no
LEFT JOIN Individual_Clients ic ON c.client_no = ic.client_no
LEFT JOIN Company_Clients cc ON c.client_no = cc.client_no;
```

Raport stanu ofert – liczba rezerwacji, uczestników, wpływy.

```
CREATE VIEW v_Offer_Status_Report AS
 WITH Activity_Price_Calc AS(
    SELECT ra.reservation_id, SUM(ra.group_size * ra.price) as activity_price
    FROM Reservation_activities ra
    GROUP BY ra.reservation_id
        )
SELECT
    o.offer_no,
    ot.name AS offer_type,
    o.offer_price,
    COUNT(DISTINCT r.reservation_id) AS total_reservations,
    SUM(r.group_size) AS total_participants,
    COALESCE(SUM((r.group_size * r.reservation_price) + ISNULL(apc.activity_price,0)), 0)  AS total_income
FROM Offers o
JOIN Offer_type ot ON o.offer_type = ot.offer_id
LEFT JOIN Reservations r ON o.offer_no = r.offer_no
LEFT JOIN Activity_Price_Calc apc ON r.reservation_id = apc.reservation_id
GROUP BY o.offer_no, ot.name, o.offer_price;
```

Lista ofert z przypisanymi aktywnościami i ich szczegółami.

```
CREATE VIEW v_Offers_Activities AS
SELECT
    o.offer_no,
    ot.name AS offer_name,
```

```
    oa.activity_id,
    oa.name AS activity_name,
    oa.max_attendees,
    oa.activity_price,
    oa.start_date,
    oa.end_date,
    oa.description
FROM Offers o
JOIN Offer_type ot ON o.offer_type = ot.offer_id
JOIN Offer_Activities oa ON o.offer_no = oa.offer_no;
```

Szczegóły uczestników i przypisane aktywności.

```
CREATE VIEW v_Participant_Activities_Details AS
SELECT
    pd.participant_id,
    pd.first_name,
    pd.last_name,
    pd.email,
    pa.activity_id,
    oa.name AS activity_name,
    oa.start_date,
    oa.end_date
FROM Participants_details pd
JOIN Participants_activities pa ON pd.participant_id = pa.participant_id
JOIN Offer_Activities oa ON pa.activity_id = oa.activity_id;
```

Widok pokazujący najczęściej używane metody płatności i jaki wpływ z nich był.

```
CREATE VIEW v_Payment_Method_Details AS
SELECT
    method,
    COUNT(*) AS payment_count,
    SUM(amount) AS total_amount
FROM Payments
GROUP BY method;
```

Widok podsumowujący rezerwację.

```
CREATE VIEW v_Reservation_Summary AS
SELECT
    c.client_no,
    COALESCE(
        (SELECT company_name FROM Company_Clients cc WHERE cc.client_no = c.client_no),
        (SELECT ic.first_name + ' ' + ic.last_name FROM Individual_Clients ic WHERE ic.client_no = c.client_no)
    ) AS client_name,
    r.reservation_id,
    r.status,
    r.group_size,
    (r.group_size * r.reservation_price) AS reservation_price_total,
    (
        SELECT COALESCE(SUM(ra.group_size * ra.price), 0)
        FROM Reservation_activities ra
        WHERE ra.reservation_id = r.reservation_id
    ) AS activities_price_total,
    (
        (r.group_size * r.reservation_price) +
        (
            SELECT COALESCE(SUM(ra.group_size * ra.price), 0)
            FROM Reservation_activities ra
            WHERE ra.reservation_id = r.reservation_id
        )
    ) AS total_price,
    (
        SELECT COALESCE(SUM(p.amount), 0)
        FROM Payments p
        WHERE p.reservation_id = r.reservation_id
    ) AS amount_paid,
```

```
    (
        (r.group_size * r.reservation_price) +
        (
            SELECT COALESCE(SUM(ra.group_size * ra.price), 0)
            FROM Reservation_activities ra
            WHERE ra.reservation_id = r.reservation_id
        )
        -
        (
            SELECT COALESCE(SUM(p.amount), 0)
            FROM Payments p
            WHERE p.reservation_id = r.reservation_id
        )
    ) AS amount_due,
    (
        SELECT ot.name
        FROM Offers o
        JOIN Offer_type ot ON o.offer_type = ot.offer_id
        WHERE o.offer_no = r.offer_no
    ) AS offer_name,
    (
        SELECT COUNT(*)
        FROM Participants_details pd
        WHERE pd.reservation_id = r.reservation_id
          AND pd.first_name IS NOT NULL
          AND pd.last_name IS NOT NULL
    ) AS filled_participants_count
FROM Reservations r
JOIN Clients c ON r.client_no = c.client_no;
```

Podwidok widoku z balansem klientów, który zwraca liste klientów oraz ich należność w przypadku nadpłaty.

```
CREATE VIEW v_Return_Money_to_Clients AS
SELECT *
FROM v_Client_Balance
WHERE amount_paid > total_price;
```

Nadchodzące aktywności z szczegółami.

```
CREATE VIEW v_Upcoming_Activities AS
SELECT
    oa.activity_id,
    oa.name AS activity_name,
    oa.start_date,
    oa.end_date,
    o.offer_no,
    ot.name AS offer_type,
    ot.Location,
    c.name AS country
FROM Offer_Activities oa
JOIN Offers o ON oa.offer_no = o.offer_no
JOIN Offer_type ot ON o.offer_type = ot.offer_id
JOIN Country c ON ot.Country = c.country_id
WHERE oa.start_date > GETDATE();
```

Procedury / funkcje

Dodaje nowego klienta firmowego do bazy danych. Wstawia dane do tabel `Clients` oraz `Company_Clients`.

```
CREATE PROCEDURE dbo.AddCompanyClient
    @CountryID      INT,
    @Address        VARCHAR(150),
    @City           VARCHAR(50),
    @Region         VARCHAR(50),
    @PostalCode     VARCHAR(15),
    @Email          VARCHAR(100),
```

```
    @PhoneNumber    VARCHAR(15),
    @Fax            VARCHAR(40)       = NULL,
    @BankNo         VARCHAR(30),
    @CompanyName    VARCHAR(200),
    @NIP            VARCHAR(12)       = NULL
AS
BEGIN
    SET NOCOUNT ON;
    BEGIN TRANSACTION;
    BEGIN TRY

        -- Wstawienie danych klienta do tabeli Clients
        INSERT INTO dbo.Clients
            (country_id, address, city, region, postal_code, email, phone_number, fax, bank_no)
        VALUES
            (@CountryID, @Address, @City, @Region, @PostalCode, @Email, @PhoneNumber, @Fax, @BankNo);

        DECLARE @NewClientNo INT;
        SET @NewClientNo = SCOPE_IDENTITY();

        -- Wstawienie danych firmowych do tabeli Company_Clients
        INSERT INTO dbo.Company_Clients
            (client_no, company_name, NIP)
        VALUES
            (@NewClientNo, @CompanyName, @NIP);

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;
        THROW;
    END CATCH
END;
```

Dodaje nowego klienta indywidualnego do bazy danych. Wstawia dane do tabel `Clients` oraz `Individual_Clients`.

```
CREATE PROCEDURE dbo.AddIndividualClient
    @CountryID      INT,
    @Address        VARCHAR(150),
    @City           VARCHAR(50),
    @Region         VARCHAR(50),
    @PostalCode     VARCHAR(15),
    @Email          VARCHAR(100),
    @PhoneNumber    VARCHAR(15),
    @Fax            VARCHAR(40)       = NULL,
    @BankNo         VARCHAR(30),
    @FirstName      VARCHAR(30),
    @LastName       VARCHAR(30)
AS
BEGIN
    SET NOCOUNT ON;
    BEGIN TRANSACTION;
    BEGIN TRY

        -- Wstawienie danych klienta do tabeli Clients
        INSERT INTO dbo.Clients
            (country_id, address, city, region, postal_code, email, phone_number, fax, bank_no)
        VALUES
            (@CountryID, @Address, @City, @Region, @PostalCode, @Email, @PhoneNumber, @Fax, @BankNo);

        DECLARE @NewClientNo INT;
        SET @NewClientNo = SCOPE_IDENTITY();

        -- Wstawienie danych indywidualnych do tabeli Individual_Clients
        INSERT INTO dbo.Individual_Clients
            (client_no, first_name, last_name)
        VALUES
            (@NewClientNo, @FirstName, @LastName);

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
```

```
            ROLLBACK TRANSACTION;
        THROW;
    END CATCH
END;
```

Dodaje nową ofertę do tabeli `Offers`.

```
CREATE PROCEDURE dbo.AddOffer
    @OfferTypeID        INT,
    @MaxParticipants    INT,
    @OfferPrice         DECIMAL(10,2),
    @StartDate          DATE,
    @EndDate            DATE,
    @DetailsDesc        VARCHAR(2000),
    @OfferActiveTime    DATE
AS
BEGIN
    SET NOCOUNT ON;

    -- Sprawdzenie dat
    IF @EndDate < @StartDate
    BEGIN
        THROW 51113, 'Data zakończenia nie może być wcześniejsza od daty rozpoczęcia.', 1;
    END

    -- Wstawianie oferty
    INSERT INTO dbo.Offers
        (offer_type, max_participants, offer_price, start_date, end_date, details_desc, offer_active_time)
    VALUES
        (@OfferTypeID, @MaxParticipants, @OfferPrice, @StartDate, @EndDate, @DetailsDesc, @OfferActiveTime);

    DECLARE @NewOfferNo         INT;
    SET @NewOfferNo = SCOPE_IDENTITY();
END;
```

Dodawanie atrakcji wycieczki do tabeli `Offer_Activity`.

```
CREATE PROCEDURE dbo.AddOffer_Activity
    @Offer_no             INT,
    @max_group            INT,
    @price                DECIMAL(10,2),
    @name                 VARCHAR(150),
    @start_date           DATETIME,
    @end_date             DATETIME,
    @description          VARCHAR(2000)
AS
BEGIN
    SET NOCOUNT ON;

    -- Sprawdzenie czy podana oferta istnieje
    IF NOT EXISTS (SELECT 1 FROM Offers WHERE offer_no = @Offer_no)
        BEGIN
            THROW 51113, 'Podana oferta nie istnieje', 1;
        END

    -- Sprawdzenie poprawności dat
    IF @end_date < @start_date
        BEGIN
            THROW 51113, 'Data zakończenia nie może być wcześniejsza od daty rozpoczęcia.', 1;
        END

    IF @start_date < (SELECT  start_date FROM Offers WHERE offer_no = @Offer_no)
        BEGIN
            THROW 51113, 'data rozpoczęcia musi być późniejsza niż data rozpoczęcia wycieczki', 1;
        END

    IF @end_date > (SELECT  end_date FROM Offers WHERE offer_no = @Offer_no)
        BEGIN
            THROW 51113, 'data zakończenia musi być wcześniejsza niż data zakończenia wycieczki', 1;
        END
```

```
    -- Sprawdzenie czy liczba miejsc w wycieczce jest mniejsza niż ta maksymalna
    IF @max_group > (SELECT  max_participants FROM Offers WHERE offer_no = @Offer_no)
        BEGIN
            THROW 51113, 'maksymalna ilość miejsc musi być mniejsza od maksymalnej ilości miejsc na wycieczkę', 1;
        END

    -- Wstawianie offerty
    INSERT INTO dbo.Offer_Activities
    (offer_no, name, max_attendees, activity_price, start_date, end_date, description)
    VALUES
        (@Offer_no, @name, @max_group, @price, @start_date,@end_date,@description);

    DECLARE @NewID INT;
    SET @NewID = SCOPE_IDENTITY();
END;
```

Dodaje nowy typ wycieczki do tabeli `Offer_type`.

```
CREATE PROCEDURE dbo.AddOffer_type
    @Country     INT,
    @Location    VARCHAR(100),
    @name        VARCHAR(150),
    @description VARCHAR(2000)
AS
    BEGIN
        SET NOCOUNT ON;
        -- Sprawdzenie czy istnieje podane państwo
        IF NOT EXISTS (SELECT 1 FROM Country where country_id = @Country)
            BEGIN
                THROW 51113, 'Nie istnieje takie państwo', 1;
            END

        -- Wstawianie typu wycieczki
        INSERT INTO dbo.Offer_type
        (Country, Location, name, description)
        VALUES
            (@Country,@Location,@name,@description);

        DECLARE @offer_id INT;
        SET @offer_id = SCOPE_IDENTITY();
    END;
```

Dodaje powiązanie uczestnika z aktywnością w tabeli `Participants_activities`.

```
CREATE PROCEDURE dbo.AddParticipantActivity
    @activity_reservation_id    INT,
    @participant_id             INT

AS
BEGIN TRY

    -- Sprawdzenie czy podany uczestnik istnieje
    IF NOT EXISTS (
        SELECT 1
        FROM Participants_details
        WHERE participant_id = @participant_id
    )
    BEGIN
        THROW 50001, 'Nie istnieje taki uczestnik!', 1;
    END

    DECLARE @participant_res_id INT;
    DECLARE @activity_res_id INT;

    SELECT @participant_res_id = pd.reservation_id
    FROM Participants_details pd
    WHERE participant_id = @participant_id;
    SELECT @activity_res_id = ra.reservation_id
    FROM Reservation_activities ra
    WHERE ra.activity_reservation_id = @activity_reservation_id
```

```
    -- Sprawdzenie czy rezerwacja atrakcji nie jest powiązana z inną rezerwacją niż uczestnik
    IF @participant_res_id <> @activity_res_id
    BEGIN
        THROW 50001, 'Rezerwacja atrakcji jest powiązana z inną rezerwacją niż uczestnik', 1;
    END

    DECLARE @reserved INT;

    SELECT @reserved = count(*)
    FROM Participants_activities pa
    WHERE activity_reservation_id = @activity_reservation_id

    -- Sprawdzenie czy wszyscy uczestnicy nie zostali już przypisani
    IF @reserved = (
        SELECT group_size
        FROM Reservation_activities
        WHERE activity_reservation_id = @activity_reservation_id
    )
    BEGIN
        THROW 50001, 'Wszyscy uczestnicy zostali już przypisani!', 1;
    END

    -- Sprawdzenie czy uczestnik nie jest już przypisany do podobnej aktywności
    IF @participant_id IN (
        SELECT participant_id
        FROM Participants_activities
        WHERE activity_reservation_id = @activity_reservation_id
    )
    BEGIN
        THROW 50001, 'Ten uczestnik jest już przypisany do podanej aktywności', 1;
    END

    -- Wstawianie
    INSERT INTO Participants_activities (
        activity_reservation_id, participant_id
    )
    VALUES (
                @activity_reservation_id,@participant_id
           );

    DECLARE @id INT;
    SET @id = SCOPE_IDENTITY();
END TRY
BEGIN CATCH
    THROW;
END CATCH;
```

Dodaje uczestnika do tabeli `Participants_details` powiązanego z rezerwacją.

```
CREATE PROCEDURE dbo.AddParticipants_details
    @Reservation_id     INT,
    @First_name         VARCHAR(50),
    @Last_name          VARCHAR(50),
    @Email              VARCHAR(50),
    @Address            VARCHAR(200)
AS
BEGIN TRY

    -- Sprawdzenie czy istnieje rezerwacja
    IF NOT EXISTS (
        SELECT 1
        FROM Reservations
        WHERE reservation_id = @Reservation_id
    )
    BEGIN
        THROW 50001, 'Nie można dodać uczestnika, który nie jest przypisany do rezerwacji!', 1;
    END

    -- Wstawienie uczestnika
    INSERT INTO Participants_details (
        reservation_id, first_name, last_name, email, address
    )
    VALUES (
        @Reservation_id, @First_name, @Last_name, @Email, @Address
```

```
    );
    DECLARE @NewParticipant_id INT;
    SET @NewParticipant_id = SCOPE_IDENTITY();
END TRY
BEGIN CATCH
    THROW;
END CATCH;
```

Procedura dodawania płatności.

```
CREATE PROCEDURE dbo.AddPayment
    @ReservationID      INT,
    @Amount             DECIMAL(10,2),
    @Method             VARCHAR(20),
    @PaymentDate        DATETIME
AS
BEGIN
    SET NOCOUNT ON;
    DECLARE @NewPaymentID INT;
    BEGIN TRANSACTION;
    BEGIN TRY

        -- Sprawdzenie czy istnieje podana rezerwacja
        IF NOT EXISTS (SELECT 1 FROM Reservations WHERE reservation_id=@ReservationID)
        BEGIN
            THROW 50001, 'Nie istnieje taka rezerwacja', 1;
        END

        -- Dodanie płatności
        INSERT INTO dbo.Payments
            (reservation_id, amount, method, date)
        VALUES
            (@ReservationID, @Amount, @Method, @PaymentDate);
        SET @NewPaymentID = SCOPE_IDENTITY();

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;
        THROW;
    END CATCH
END;
```

Tworzy nową rezerwację dla klienta na wybraną ofertę.

```
CREATE PROCEDURE dbo.AddReservation
    @client_no          INT,
    @offer_no           INT,
    @group_size         INT
AS
BEGIN
    BEGIN TRANSACTION;
    BEGIN TRY

        -- Sprawdzenie czy podany klient istnieje
        IF NOT EXISTS (
            SELECT 1 FROM Clients
            WHERE client_no = @client_no
        )
        BEGIN
            THROW 50001, 'Błąd – nie ma takiego klienta', 1;
        END

        -- Sprawdzenie czy podana oferta istnieje
        IF NOT EXISTS (
            SELECT 1 FROM Offers
            WHERE offer_no = @offer_no
        )
            BEGIN
                THROW 50002, 'Błąd – nie ma takiej oferty', 1;
```

```
            END

        DECLARE @reservation_price DECIMAL(10,2);
        SELECT @reservation_price = offer_price FROM Offers WHERE offer_no = @offer_no;

        -- Wstawianie
        INSERT INTO Reservations(
            client_no, offer_no, group_size, reservation_price, status
        )
        VALUES (
                   @client_no, @offer_no, @group_size, @reservation_price, 'D'
               );

        DECLARE @Reservation_id INT;
        SET @Reservation_id = SCOPE_IDENTITY();

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;
        THROW;
    END CATCH
END;
```

Dodaje aktywność do rezerwacji w tabeli `Reservation_activities`.

```
CREATE PROCEDURE dbo.AddReservationActivity
    @Reservation_id     INT,
    @Activity_id        INT,
    @group_size         INT
AS
BEGIN
    BEGIN TRANSACTION;
    BEGIN TRY
        IF NOT EXISTS (
            SELECT 1 FROM Reservations
            WHERE reservation_id = @Reservation_id
        )
        BEGIN
            THROW 50001, 'Błąd – nie ma takiego reservation_id', 1;
        END

        IF NOT EXISTS (
            SELECT 1 FROM Offer_Activities
            WHERE activity_id = @Activity_id
        )
        BEGIN
            THROW 50002, 'Błąd – nie ma takiego activity_id', 1;
        END
        DECLARE @price DECIMAL(10,2);
        SELECT @price = activity_price FROM Offer_Activities WHERE activity_id = @Activity_id;
        INSERT INTO Reservation_activities (
            reservation_id, activity_id, group_size, price
        )
        VALUES (
            @Reservation_id, @Activity_id, @group_size, @Price
        );

        DECLARE @NewActivity_Reservation_id INT;
        SET @NewActivity_Reservation_id = SCOPE_IDENTITY();

        COMMIT TRANSACTION;
    END TRY

    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;
        THROW;
    END CATCH
END;
```

Pozwala pracownikowi anulować dowolną rezerwację.

```
CREATE PROCEDURE dbo.CancelByEmployee
    @Reservation_id     INT
AS
BEGIN
    BEGIN TRANSACTION;
    BEGIN TRY

        -- Sprawdzenie czy podana rezerwacja istnieje
        IF NOT EXISTS (
            SELECT 1
            FROM Reservations
            WHERE reservation_id = @reservation_id
        )
        BEGIN
            THROW 50002, 'Błąd – nie ma takiej rezerwacji', 1;
        END

        -- Aktualizacja statusu rezerwacji
        UPDATE r
        SET r.status = 'A'
        FROM Reservations r WHERE reservation_id = @reservation_id;

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;
        THROW;
    END CATCH
END;
```

Umożliwia klientowi anulowanie własnej rezerwacji.

```
CREATE PROCEDURE dbo.CancelByUser
    @Reservation_id     INT,
    @client_no          INT
AS
BEGIN
    BEGIN TRANSACTION;
    BEGIN TRY

        -- Sprawdzenie czy klient i rezerwacja istnieją
        IF NOT EXISTS (
            SELECT 1
            FROM Clients
            WHERE client_no = @client_no
        )
        BEGIN
            THROW 50001, 'Błąd – nie ma takiego klienta', 1;
        END

        IF NOT EXISTS (
            SELECT 1
            FROM Reservations
            WHERE reservation_id = @reservation_id
        )
        BEGIN
            THROW 50002, 'Błąd – nie ma takiej rezerwacji', 1;
        END

        DECLARE @reserv_cli INT;
        SELECT @reserv_cli = client_no FROM Reservations WHERE reservation_id = @reservation_id;

        IF @reserv_cli <> @client_no
            BEGIN
                THROW 50002, 'Błąd – rezerwacja nie należy do tego klienta', 1;
            END

        -- Aktualizacja statusu rezerwacji
        UPDATE r
        SET r.status = 'K'
```

```
        FROM Reservations r WHERE reservation_id = @reservation_id;

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;
        THROW;
    END CATCH
END;
```

Ustawia status rezerwacji na 'A' jeśli nie podano wszystkich danych uczestników wycieczki.

```
CREATE PROCEDURE dbo.CancelReservationNoParticipantData
AS
BEGIN

    -- Anulowanie rezerwacji
    UPDATE R
    SET R.status = 'A'
    FROM Reservations R
    JOIN Offers O ON R.offer_no = O.offer_no
    JOIN (
        SELECT COUNT(*) registered, P.reservation_id
        FROM Participants_details P
        GROUP BY P.reservation_id
    ) pd on pd.reservation_id = R.reservation_id
    WHERE
        R.status IN ('D','Z')
        AND DATEDIFF(DAY, GETDATE(), O.start_date) <= 7
        AND COALESCE(pd.registered,0) <> R.group_size
END;
```

Aktualizuje status rezerwacji na 'N' jeśli termin płatności minął.

```
CREATE PROCEDURE dbo.CancelUnpaidReservations
AS
BEGIN

    -- Anulowanie rezerwacji
    UPDATE R
    SET R.status = 'N'
    FROM Reservations R
    JOIN Offers O ON R.offer_no = O.offer_no
    WHERE
        R.status = 'D'
        AND DATEDIFF(DAY, GETDATE(), O.start_date) <= 7;
END;
```

Pozwala zmienić rozmiar grupy dla danej aktywności.

```
CREATE PROCEDURE dbo.EditActivitiReservation
        @activity_reservation_id    INT,
        @new_group_size             INT
AS
BEGIN
    BEGIN TRANSACTION;
    BEGIN TRY

        -- Sprawdzenie czy istnieje taka rezerwacja aktywności
        IF NOT EXISTS (
            SELECT 1 FROM Reservation_activities WHERE activity_reservation_id = @activity_reservation_id
        )
        BEGIN
            THROW 50002, 'Błąd – nie istnieje taka rezerwacja aktywności', 1;
        END

        -- Sprawdzenie ilości uczestników (czy nie za dużo)
```

```
        DECLARE @participant_size INT;
        SELECT @participant_size = COUNT (*)
        FROM Participants_activities
        WHERE activity_reservation_id = @activity_reservation_id;

        IF @participant_size > @new_group_size
            BEGIN
                THROW 50002, 'Błąd – do aktywności jest przypisana za duża ilość uczestników', 1;
            END

        -- Zmiana wielkości grupy
        UPDATE ra
        SET ra.group_size = @new_group_size
        FROM Reservation_activities ra
        WHERE ra.activity_reservation_id = @activity_reservation_id;

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;
        THROW;
    END CATCH
END;
```

Pozwala edytować dane uczestnika wycieczki.

```
CREATE PROCEDURE dbo.EditParticipant
    @participant_id     INT,
    @First_name         VARCHAR(50) = NULL,
    @Last_name          VARCHAR(50) = NULL,
    @Email              VARCHAR(50) = NULL,
    @Address            VARCHAR(200) = NULL
AS
BEGIN
    BEGIN TRANSACTION;
    BEGIN TRY

        -- Sprawdzenie czy istnieje taki uczestnik
        IF NOT EXISTS (
            SELECT 1 FROM Participants_details WHERE participant_id = @participant_id
        )
        BEGIN
            THROW 50002, 'Błąd – nie istnieje taki uczestnik', 1;
        END

        -- Edycja danych uczestnika wycieczki
        UPDATE pd
        SET     pd.first_name = COALESCE(@First_name, pd.first_name),
                pd.last_name = COALESCE(@Last_Name, pd.last_name),
                pd.email = COALESCE(@Email, pd.email),
                pd.address = COALESCE(@Address, pd.address)
        FROM Participants_details pd
        WHERE pd.participant_id = @participant_id;

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;
        THROW;
    END CATCH
END;
```

Pozwala zmienić uczestnika lub aktywność w powiązaniu.

```
CREATE PROCEDURE dbo.EditParticipantActivity
        @id                         INT,
        @participant_id             INT,
        @new_reservation_activity   INT
AS
```

```
BEGIN
    BEGIN TRANSACTION;
    BEGIN TRY

        -- Sprawdzenie czy dane przypisanie uczestnika do wycieczki istnieje
        IF NOT EXISTS (
            SELECT 1 FROM Participants_activities
            WHERE id = @id
        )
        BEGIN
            THROW 50002, 'Błąd – nie istnieje takie przypisanie uczestnika do wycieczki', 1;
        END

        -- Sprawdzenie czy istnieje podany uczestnik
        IF NOT EXISTS (
                SELECT 1 FROM Participants_details
                WHERE  participant_id = @participant_id
        )
        BEGIN
            THROW 50002, 'Błąd – nie istnieje taki uczestnik', 1;
        END

        -- Sprawdzenie czy istnieje podana rezeracja
        IF NOT EXISTS (
            SELECT 1 FROM Reservation_activities
            WHERE activity_reservation_id = @new_reservation_activity
        )
        BEGIN
            THROW 50002, 'Błąd – nie istnieje taka rezerwacja aktywności', 1;
        END

        DECLARE @old_participant_res_id  INT;
        DECLARE @new_participant_res_id  INT;
        DECLARE @old_participant_id      INT;

        SELECT @old_participant_id = pa.participant_id FROM Participants_activities pa WHERE id = @id;
        SELECT @old_participant_res_id = pd.reservation_id FROM Participants_details pd WHERE participant_id =
@old_participant_id;
        SELECT @new_participant_res_id = pd.reservation_id FROM Participants_details pd WHERE participant_id =
@participant_id;

        DECLARE @old_participant_res_id  INT;
        DECLARE @new_participant_res_id  INT;
        DECLARE @old_participant_id      INT;

        SELECT @old_participant_id = pa.participant_id FROM Participants_activities pa WHERE id = @id;
        SELECT @old_participant_res_id = pd.reservation_id FROM Participants_details pd WHERE participant_id =
@old_participant_id;
        SELECT @new_participant_res_id = pd.reservation_id FROM Participants_details pd WHERE participant_id =
@participant_id;

        -- Sprawdzenie czy uczestnicy nie należą do tych samych wycieczek
        IF @old_participant_res_id <> @new_participant_res_id
        BEGIN
            THROW 50002, 'Błąd - uczestnicy należą do różnych wycieczek ', 1;
        END

        -- Zmiana
        UPDATE pa
        SET pa.participant_id = @participant_id, pa.activity_reservation_id = @new_reservation_activity
        FROM Participants_activities pa
        WHERE pa.id = @id;

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;
        THROW;
    END CATCH
END;
```

Pozwala na edycję zamówienia (zmiana liczby uczestników).

```
CREATE PROCEDURE dbo.EditReservation
    @reservation_id INT,
    @group_size INT
AS
BEGIN
    BEGIN TRANSACTION;
    BEGIN TRY

        -- Sprawdzenie czy istnieje podana rezerwacja
        IF NOT EXISTS (
            SELECT 1 FROM Reservations r
            WHERE  r.reservation_id = @reservation_id
        )
        BEGIN
            THROW 50002, 'Błąd – nie istnieje taka rezerwacja', 1;
        END

        DECLARE  @participant_count INT;
        SELECT @participant_count = COUNT(*)
        FROM Participants_details
        WHERE reservation_id = @reservation_id;

        IF @participant_count > @group_size
        BEGIN
            THROW 50002, 'Błąd – ilość uczestników powiązanych z rezerwacją jest większa niż nowy rozmiar grupy', 1;
        END

        DECLARE @max_activity_size INT;
        SELECT @max_activity_size = MAX(group_size) FROM Reservation_activities
        WHERE reservation_id = @reservation_id;

        IF @max_activity_size > @group_size
            BEGIN
                THROW 50002, 'Błąd – jedna z aktywności ma większy rozmiar niż nowy rozmiar grupy', 1;
            END

        -- Zmiana liczby uczestników
        UPDATE r
        SET r.group_size = @group_size
        FROM Reservations r
        WHERE r.reservation_id = @reservation_id;

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;
        THROW;
    END CATCH
END;
```

Usuwa rezerwację aktywności oraz powiązane przypisania uczestników.

```
CREATE PROCEDURE dbo.RemoveActivitiReservation
    @activity_reservation_id INT
AS
BEGIN
    BEGIN TRANSACTION;
    BEGIN TRY

        -- Sprawdzenie czy istnieje taka rezerwacja aktywności
        IF NOT EXISTS (
            SELECT 1 FROM Reservation_activities WHERE activity_reservation_id = @activity_reservation_id
        )
        BEGIN
            THROW 50002, 'Błąd – nie istnieje taka rezerwacja aktywności', 1;
        END

        -- Usuwanie rezerwacji aktywności
        DELETE FROM Participants_activities WHERE activity_reservation_id = @activity_reservation_id;
        DELETE FROM Reservation_activities  WHERE activity_reservation_id = @activity_reservation_id;

        COMMIT TRANSACTION;
```

```
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;
        THROW;
    END CATCH
END;
```

Usuwa uczestnika oraz wszystkie powiązania z aktywnościami.

```
CREATE PROCEDURE dbo.RemoveParticipant
    @participant_id INT
AS
BEGIN
    BEGIN TRANSACTION;
    BEGIN TRY

        -- Sprawdzenie czy podany uczestnik aktywności istnieje
        IF NOT EXISTS (
            SELECT 1 FROM Participants_details pd
            WHERE pd.participant_id = @participant_id
        )
        BEGIN
            THROW 50002, 'Błąd – podany uczestnik nie istnieje', 1;
        END

        -- Usuwanie uczestnika
        DELETE FROM Participants_activities WHERE participant_id = @participant_id;
        DELETE FROM Participants_details WHERE participant_id = @participant_id;

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;
        THROW;
    END CATCH
END;
```

Usuwa powiązanie uczestnika z aktywnością.

```
CREATE PROCEDURE dbo.RemoveParticipant_activity
    @participant_id             INT,
    @activity_reservation_id    INT
AS
BEGIN
    BEGIN TRANSACTION;
    BEGIN TRY

        -- Sprawdzenie czy uczestnik był przypisany do danej aktywności
        IF NOT EXISTS (
            SELECT 1 FROM Participants_activities pa
            WHERE pa.participant_id = @participant_id AND pa.activity_reservation_id = @activity_reservation_id
        )
        BEGIN
            THROW 50002, 'Błąd – uczestnik nie jest przypisane do danej aktywności', 1;
        END

        -- Usuwanie powiązania uczestnika z aktywnością
        DELETE FROM Participants_activities WHERE participant_id = @participant_id AND activity_reservation_id =
@activity_reservation_id;

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;
        THROW;
    END CATCH
END;
```

Historia wszystkich aktywności, w których uczestniczył dany klient (kupił je).

```
CREATE FUNCTION dbo.z_fn_ClientActivityHistory (@ClientID INT)
RETURNS TABLE
AS
RETURN
(
    SELECT
        ra.activity_id,
        oa.name AS activity_name,
        oa.start_date,
        oa.end_date,
        oa.activity_price
    FROM Reservations r
    JOIN Reservation_activities ra ON r.reservation_id = ra.reservation_id
    JOIN Offer_Activities oa ON ra.activity_id = oa.activity_id
    WHERE r.client_no = @ClientID
);
```

Funkcja `fn_ClientReservations` generuje zestawienie wszystkich rezerwacji dokonanych przez klienta o podanym `client_no'.

```
CREATE FUNCTION z_fn_ClientReservations (@ClientID INT)
RETURNS TABLE
AS
RETURN
(
    SELECT
        r.reservation_id,
        r.status,
        r.group_size,
        r.reservation_price,
        o.offer_no,
        ot.name AS offer_type_name,
        o.offer_price,
        o.start_date,
        o.end_date,
        COALESCE(SUM(p.amount), 0) AS total_paid
    FROM Reservations r
    JOIN Offers o ON r.offer_no = o.offer_no
    JOIN Offer_type ot ON o.offer_type = ot.offer_id
    LEFT JOIN Payments p ON r.reservation_id = p.reservation_id
    WHERE r.client_no = @ClientID
    GROUP BY
        r.reservation_id, r.status, r.group_size, r.reservation_price,
        o.offer_no, ot.name, o.offer_price, o.start_date, o.end_date
);
```

Zwraca sumaryczne informacje o płatnościach klienta – łączna kwota, liczba płatności, średnia wpłata.

```
CREATE FUNCTION dbo.z_fn_ClientPaymentSummary (@ClientID INT)
RETURNS TABLE
AS
RETURN
(
    SELECT
        c.client_no,
        COUNT(p.payment_id) AS payment_count,
        SUM(p.amount) AS total_paid,
        AVG(p.amount) AS avg_payment
    FROM Clients c
    JOIN Reservations r ON c.client_no = r.client_no
    JOIN Payments p ON r.reservation_id = p.reservation_id
    WHERE c.client_no = @ClientID
    GROUP BY c.client_no
);
```

Triggery

Trigger automatycznie ustawia status rezerwacji na "Z", jeśli suma wpłat pokrywa koszt rezerwacji oraz aktywności.

```
CREATE TRIGGER increase_status_based_on_amount
ON Payments
AFTER INSERT
AS
BEGIN

    -- Pobranie ceny rezerwacji
    DECLARE @reservation_price DECIMAL(10,2);
    SELECT @reservation_price = r.group_size * r.reservation_price
    FROM Reservations r
    JOIN inserted i ON i.reservation_id = r.reservation_id;

    -- Pobranie ceny aktywności
    DECLARE @activities_price DECIMAL(10,2);
    SELECT  @activities_price = SUM(ra.group_size*ra.price)
    FROM Reservation_activities ra
    JOIN inserted i ON i.reservation_id = ra.reservation_id;

    -- Zaktualizowanie status rezerwacji na "Z", jeśli suma wpłat >= koszt rezerwacji + aktywności
    UPDATE R
    SET status = 'Z'
    FROM Reservations R
    JOIN inserted on inserted.reservation_id = R.reservation_id
    JOIN (
        SELECT reservation_id, SUM(amount) AS total_paid
        FROM Payments P
        GROUP BY reservation_id
    ) P ON R.reservation_id = P.reservation_id
    WHERE P.total_paid >= @reservation_price + @activities_price;
END;
```

Sprawdzanie czy jest wystarczająca ilość miejsc dla grupy na wycieczce oraz czy można jeszcze edytować zamówienie - aktywowane podczas dodawania nowej rezerwacji oraz podczas edycji już istniejącej rezerwacji (nasza baza nie przewiduje usuwania rekordów z tabeli reservations, tylko zmianę statusu).

```
CREATE TRIGGER validate_reservation
ON Reservations
INSTEAD OF INSERT, UPDATE
AS
BEGIN
    DECLARE @reservation_id INT;
    SELECT @reservation_id = reservation_id FROM inserted;

    DECLARE @offer_no INT;
    SELECT @offer_no = offer_no FROM inserted;

    -- Sprawdzenie czy można edytować rezerwacje
    DECLARE @start_date DATE;
    SELECT @start_date = o.start_date
    FROM Offers o
    WHERE o.offer_no = @offer_no;

    IF DATEDIFF(DAY, GETDATE(), @start_date) <= 7
    BEGIN
        THROW 50002, 'Błąd – nie można już edytować rezerwacji', 1;
    END

    DECLARE @taken INT;
    DECLARE @max_participants INT;

    SELECT @taken = SUM(r.group_size)
    FROM Reservations r
    WHERE r.status NOT IN ('A', 'K')
      AND r.offer_no = @offer_no
      AND r.reservation_id <> @reservation_id;

    SELECT @max_participants = o.max_participants
    FROM Offers o
```

```
    WHERE o.offer_no = @offer_no;

    DECLARE @new_group_size INT;

    IF NOT EXISTS (SELECT 1 FROM deleted) -- check if insert
    BEGIN
        SELECT @new_group_size = SUM(group_size) FROM inserted;
        IF (@taken + @new_group_size > @max_participants)
        BEGIN
            THROW 50001, 'Błąd – rozmiar grupy jest za duży', 1;
            ROLLBACK TRANSACTION;
            RETURN;
        END
        ELSE
        BEGIN
            -- If not exceeding capacity, insert the records
            INSERT INTO Reservations (
                client_no, offer_no, group_size, status, reservation_price
            )
            SELECT
                client_no, offer_no, group_size, status, reservation_price
            FROM inserted;
        END
    END
    ELSE -- handle update
    BEGIN
        SELECT @new_group_size = COALESCE(SUM(group_size), 0) FROM inserted;

        DECLARE @status_pre CHAR(1);
        SELECT @status_pre = status FROM deleted;

        DECLARE @status_post CHAR(1);
        SELECT @status_post = status FROM inserted;

        IF @status_pre = @status_post
        BEGIN
            IF (@taken + @new_group_size > @max_participants)
            BEGIN
                THROW 50001, 'Błąd – rozmiar nowej grupy jest za duży', 1;
                ROLLBACK TRANSACTION;
                RETURN;
            END
        END

        UPDATE r
        SET r.group_size = i.group_size,
            r.status = i.status
        FROM Reservations r
        JOIN inserted i ON i.reservation_id = r.reservation_id;
    END
END;
```

Sprawdzanie czy jest możliwe dokonie edycji rezerwacji aktywności - sprawdza czy nie minął ostateczny termin na zmiany oraz czy jest odpowiednia ilość miejsc w przypadku dodania nowej rezerwacji oraz w przypadku edycji istniejącej rezerwacji.

```
CREATE TRIGGER validate_activity_reservation_change
ON Reservation_activities
INSTEAD OF INSERT, UPDATE, DELETE
AS
BEGIN
    DECLARE @current_count INT,
            @max_attendee INT,
            @activity_id INT,
            @reservation_id INT,
            @group_size INT,
            @price DECIMAL(10,2),
            @start_date DATE;

    -- Sprawdzenie czy można edytować rezerwacje
    IF EXISTS (SELECT 1 FROM inserted)
    BEGIN
        DECLARE @activity_reservation_id INT;
        SELECT
            @activity_reservation_id = activity_reservation_id,
```

```
            @activity_id = activity_id,
            @reservation_id = reservation_id,
            @group_size = group_size,
            @price = price
        FROM inserted;

        SELECT DISTINCT @start_date = o.start_date
        FROM Reservations r
        JOIN Offers o ON o.offer_no = r.offer_no
        WHERE r.reservation_id = @reservation_id;

        IF DATEDIFF(DAY, GETDATE(), @start_date) <= 7
        BEGIN
            THROW 50002, 'Błąd – nie można już edytować rezerwacji', 1;
        END

        -- Sprawdzenie ilości osób
        DECLARE @res_group_size INT;
        SELECT @res_group_size = Reservations.group_size
        FROM Reservations
        WHERE Reservations.reservation_id = @reservation_id;

        IF @group_size > @res_group_size
        BEGIN
            THROW 50002, 'Błąd - ilość osób przekracza ilość zapisaną na wycieczkę', 1;
        END

        -- Sprawdzenie dostępności miejsc
        SELECT @current_count = SUM(ra.group_size)
        FROM Reservation_activities ra
        JOIN Reservations r ON r.reservation_id = ra.reservation_id AND r.status NOT IN ('A', 'K')
        WHERE activity_id = @activity_id AND ra.activity_reservation_id <> @activity_reservation_id
        GROUP BY activity_id;

        IF NOT EXISTS (SELECT 1 FROM deleted) -- INSERT HANDLE
        BEGIN
            SELECT @max_attendee = max_attendees
            FROM Offer_Activities
            WHERE activity_id = @activity_id;

            IF (@current_count + @group_size > @max_attendee)
            BEGIN
                THROW 50001, 'Brak dostępnych miejsc na aktywność', 1;
                RETURN;
            END
            ELSE
            BEGIN
                INSERT INTO Reservation_activities (reservation_id, activity_id, group_size, price)
                SELECT reservation_id, activity_id, group_size, price FROM inserted;
            END
        END
        ELSE -- UPDATE HANDLE
        BEGIN
            IF (@current_count + @group_size > @max_attendee)
            BEGIN
                THROW 50001, 'Brak dostępnych miejsc dla nowej grupy', 1;
                RETURN;
            END
            ELSE
            BEGIN
                UPDATE ra
                SET ra.group_size = @group_size
                FROM Reservation_activities ra
                JOIN inserted i ON ra.activity_reservation_id = i.activity_reservation_id;
            END
        END
    END
    ELSE -- DELETION HANDLE
    BEGIN
        SELECT @reservation_id = reservation_id FROM deleted;

        SELECT DISTINCT @start_date = o.start_date
        FROM Reservation_activities ra
        JOIN Reservations r ON r.reservation_id = ra.reservation_id
        JOIN Offers o ON o.offer_no = r.offer_no
        WHERE r.reservation_id = @reservation_id;
```

```
        IF DATEDIFF(DAY, GETDATE(), @start_date) <= 7
        BEGIN
            THROW 50002, 'Błąd – nie można już usunąć rezerwacji', 1;
        END

        DELETE Reservation_activities WHERE reservation_id = @reservation_id;
    END
END;
```

Trigger sprawdza, czy można jeszcze edytować dane uczestników (dodawać, usuwać, zmieniać).

```
CREATE TRIGGER validate_date_participants_details
ON Participants_details
FOR INSERT,DELETE,UPDATE
AS
BEGIN
    DECLARE @participant_id INT;
    IF EXISTS (SELECT 1 FROM inserted)
    BEGIN
        SELECT @participant_id = participant_id
        FROM inserted
    END
    ELSE
    BEGIN
        SELECT @participant_id = participant_id
        FROM deleted
    END

    -- Pobranie daty rozpoczęcia wycieczki
    DECLARE @start_date DATE;
    SELECT @start_date = o.start_date
    FROM Participants_details pd
    JOIN Reservations r ON r.reservation_id = pd.reservation_id
    JOIN Offers o ON o.offer_no = r.offer_no
    WHERE pd.participant_id = @participant_id

    -- Sprawdzenie, czy można jeszcze edytować dane uczestnika
    IF DATEDIFF(DAY, GETDATE(), @start_date) <= 7
    BEGIN
        THROW 50002, 'Błąd – nie można już edytować danych uczestników do tej wycieczki', 1;
    END
END;
```

Trigger sprawdza, czy można jeszcze edytować przypisania uczestników do atrakcji.

```
CREATE TRIGGER validate_date_participants_activities
ON Participants_activities
FOR INSERT, DELETE, UPDATE
AS
BEGIN
    DECLARE @participant_id INT;
    IF EXISTS (SELECT 1 FROM inserted)
    BEGIN
        SELECT @participant_id = participant_id FROM inserted
    END
    ELSE
    BEGIN
        SELECT @participant_id = participant_id FROM deleted
    END

    -- Pobieranie daty rozpoczęcia wycieczki
    DECLARE @start_date DATE;
    SELECT @start_date = o.start_date
    FROM Participants_details pd
    JOIN Reservations r ON r.reservation_id = pd.reservation_id
    JOIN Offers o ON o.offer_no = r.offer_no
    WHERE pd.participant_id = @participant_id;

    -- Sprawdzenie, czy można jeszcze edytować przypisania do atrakcji
    IF DATEDIFF(DAY, GETDATE(), @start_date) <= 7
    BEGIN
```

```
        THROW 50002, 'Błąd – nie można już edytować przypisań do atrakcji dla uczestników do tej wycieczki', 1;
    END
END;
```

---

4. Tworzenie ról:

Sprzedawca oraz osoba mająca kontakt z klientem

Rola przeznaczona dla pracowników mających bezpośredni kontakt z klientem, w tym sprzedawców oraz osób odpowiedzialnych za obsługę rezerwacji. Użytkownicy przypisani do tej roli mają możliwość przeglądania, dodawania oraz edytowania danych klientów, rezerwacji, uczestników oraz płatności. Mają również dostęp do podstawowych informacji o ofercie oraz przypisanych do niej aktywnościach.

```
CREATE ROLE Customer_Service;
```

```
-- Uprawnienia do Tabel
GRANT SELECT, INSERT, UPDATE ON Clients TO Customer_Service;
GRANT SELECT, INSERT, UPDATE ON Individual_Clients TO Customer_Service;
GRANT SELECT, INSERT, UPDATE ON Company_Clients TO Customer_Service;
GRANT SELECT, INSERT, UPDATE ON Reservations TO Customer_Service;
GRANT SELECT, INSERT, UPDATE ON Participants_details TO Customer_Service;
GRANT SELECT, INSERT, UPDATE ON Participants_activities TO Customer_Service;
GRANT SELECT, INSERT, UPDATE ON Reservation_activities TO Customer_Service;
GRANT SELECT, INSERT, UPDATE ON Payments TO Customer_Service;
GRANT SELECT ON Offers TO Customer_Service;
GRANT SELECT ON Offer_Activities TO Customer_Service;
GRANT SELECT ON Offer_type TO Customer_Service;

-- Uprawnienia do Widoków
GRANT SELECT ON v_Reservation_Summary TO Customer_Service;
GRANT SELECT ON v_Upcoming_Activities TO Customer_Service;
GRANT SELECT ON v_Available_Offers TO Customer_Service;
GRANT SELECT ON v_Client_and_Participants_Full_Reservation_Info TO Customer_Service;
GRANT SELECT ON v_Client_and_Participants_info TO Customer_Service;
GRANT SELECT ON v_Client_Reservations TO Customer_Service;
GRANT SELECT ON v_Offers_Activities TO Customer_Service;
GRANT SELECT ON v_Client_Balance_on_reservations TO Customer_Service;
GRANT SELECT ON v_Participant_Activities_Details TO Customer_Service;

-- Uprawnienia do Procedur
GRANT EXECUTE ON AddCompanyClient TO Customer_Service;
GRANT EXECUTE ON AddIndividualClient TO Customer_Service;
GRANT EXECUTE ON AddReservation TO Customer_Service;
GRANT EXECUTE ON AddReservationActivity TO Customer_Service;
GRANT EXECUTE ON AddParticipants_details TO Customer_Service;
GRANT EXECUTE ON AddParticipantActivity TO Customer_Service;
GRANT EXECUTE ON EditReservation TO Customer_Service;
GRANT EXECUTE ON EditActivitiReservation TO Customer_Service;
GRANT EXECUTE ON EditParticipant TO Customer_Service;
GRANT EXECUTE ON EditParticipantActivity TO Customer_Service;
GRANT EXECUTE ON RemoveActivitiReservation TO Customer_Service;
GRANT EXECUTE ON RemoveParticipant TO Customer_Service;
GRANT EXECUTE ON RemoveParticipant_activity TO Customer_Service;
GRANT EXECUTE ON CancelByEmployee TO Customer_Service;
GRANT EXECUTE ON AddPayment TO Customer_Service;
```

Manager, którego zadaniem jest zarządzanie ofertami oraz aktywnościami.

Rola dedykowana menedżerom odpowiedzialnym za tworzenie i zarządzanie ofertami oraz związanymi z nimi aktywnościami. Użytkownicy tej roli mogą przeglądać, dodawać, modyfikować i usuwać dane dotyczące ofert, typów ofert oraz aktywności.

```
CREATE ROLE Offer_Manager;
```

```
-- Uprawnienia do Tabel
GRANT SELECT, INSERT, UPDATE, DELETE ON Offers TO Offer_Manager;
```

```
GRANT SELECT, INSERT, UPDATE, DELETE ON Offer_type TO Offer_Manager;
GRANT SELECT, INSERT, UPDATE, DELETE ON Offer_Activities TO Offer_Manager;
GRANT SELECT, INSERT, UPDATE, DELETE ON Reservations_activities TO Offer_Manager;
GRANT SELECT ON Reservations TO Offer_Manager;

-- Uprawnienia do Widoków
GRANT SELECT ON v_Upcoming_Activities TO Offer_Manager;
GRANT SELECT ON v_Available_Offers TO Offer_Manager;
GRANT SELECT ON v_Offer_Status_Report TO Offer_Manager;
GRANT SELECT ON v_Activity_Status_Report TO Offer_Manager;
GRANT SELECT ON v_Offers_Activities TO Offer_Manager;

-- Uprawnienia do Procedur
GRANT EXECUTE ON AddOffer TO Offer_Manager;
GRANT EXECUTE ON AddOffer_type TO Offer_Manager;
GRANT EXECUTE ON AddOffer_Activity TO Offer_Manager;
```

Księgowy

Rola przeznaczona dla działu księgowości. Użytkownicy przypisani do tej roli mają wyłącznie uprawnienia do odczytu danych niezbędnych do rozliczeń finansowych, w tym informacji o klientach, rezerwacjach, płatnościach i powiązanych aktywnościach.

```
CREATE ROLE Accountant;
```

```
-- Uprawnienia do Tabel
GRANT SELECT ON Clients TO Accountant;
GRANT SELECT ON Company_Clients TO Accountant;
GRANT SELECT ON Individual_Clients TO Accountant;
GRANT SELECT ON Payments TO Accountant;

-- Uprawnienia do Widoków
GRANT SELECT ON v_Client_Balance TO Accountant;
GRANT SELECT ON v_Client_Balance_on_reservations TO Accountant;
GRANT SELECT ON v_Return_Money_to_Clients TO Accountant;
GRANT SELECT ON v_Payment_Method_Details TO Accountant;
```

Administrator bazy danych

Użytkownik Administrator z uprawnieniami SELECT, VIEW ANY DATABASE, BACKUP DATABASE oraz ADMINISTER BULK OPERATIONS może przeglądać dane w bazach, do których ma dostęp, widzieć listę wszystkich baz danych na serwerze, wykonywać kopie zapasowe oraz korzystać z operacji masowego importu i eksportu danych.

```
CREATE ROLE Administrator;
```

```
GRANT SELECT, VIEW ANY DATABASE, BACKUP DATABASE, ADMINISTER BULK OPERATIONS TO Administrator;
```

Klient bazy danych

Uprawnienia dla klientów są definiowane na poziomie warstwy aplikacyjnej. Każdy klient otrzymuje indywidualne konto, którego login tworzony jest jako połączenie imienia, nazwiska oraz ciągu cyfr. Hasło generowane jest automatycznie i ma długość 12 znaków.

5. Indexy

Index na `(reservation_id, date)` w tabeli `Payments`

```
CREATE INDEX i_t_Payments ON Payments (reservation_id, date);
```

Index na `(client_no, offer_no)` w tabeli `Reservations`

```
CREATE INDEX i_t_Reservations ON Reservations (client_no, offer_no);
```

Index na `(reservation_id, activity_id)` **w tabeli** `Reservation_activities`

```
CREATE INDEX i_t_Reservation_activities ON Reservation_activities (reservation_id, activity_id);
```

Index na `(activity_reservation_id, participant_id)` **w tabeli** `Participants_activities`

```
CREATE INDEX i_t_Participants_activities ON Participants_activities (activity_reservation_id, participant_id);
```

Index na `(reservation_id)` **w tabeli** `Participants_details`

```
CREATE INDEX i_t_Participants_details ON Participants_details (reservation_id);
```

Index na `(offer_no)` **w tabeli** `Offer_Activities`

```
CREATE INDEX i_t_Offer_Activities ON Offer_Activities (offer_no);
```

Index na `(offer_type, start_date, offer_active_time)` **w tabeli** `Offers`

```
CREATE INDEX i_t_Offers ON Offers (offer_type, start_date, offer_active_time);
```

Index na `(Country)` **w tabeli** `Offer_type`

```
CREATE INDEX i_t_Offer_type ON Offer_type (Country);
```
