

Database of a company providing tourist services (sale of trips) for individual clients and companies

1. System Requirements and Features

List of System Requirements:

- The system must enable customers (both individuals and companies) to book a tour.
- Customers can book for multiple people (tour participants).
- Customers can select additional attractions assigned to a given offer.
- Additional services cannot be booked without first booking a tour.
- When making a reservation, customers can only provide the number of tour participants and any additional services ordered.
- A week before departure, customers must provide identifying information for tour participants and attractions (cancel the order if missing).
- Customers can make changes to their orders up to 7 days before departure.
- Customers must pay for their orders up to 7 days before departure (cancel the order if missing).
- The system records payments associated with reservations (who paid, when, how much, and for what).

System User Permissions:

1. System Administrator:

- Any database modification

2. Customer:

- Browse available tour offers,
- Make tour reservations,
- Select additional attractions assigned to the tour,
- Add participants to the reservation,
- Make payments,
- Cancel a tour or activity reservation,
- View your reservations and their status,
- Modify reservation details (up to 7 days before departure).

3. Employee:

- Add and modify tour offers,
- Assign attractions to offers,
- Assign hotels, agencies, and transportation options to offers,
- View all reservations and participants,
- Generate reports (booking statistics, payments, offer popularity),
- Cancel or restore reservations manually.

User Stories:

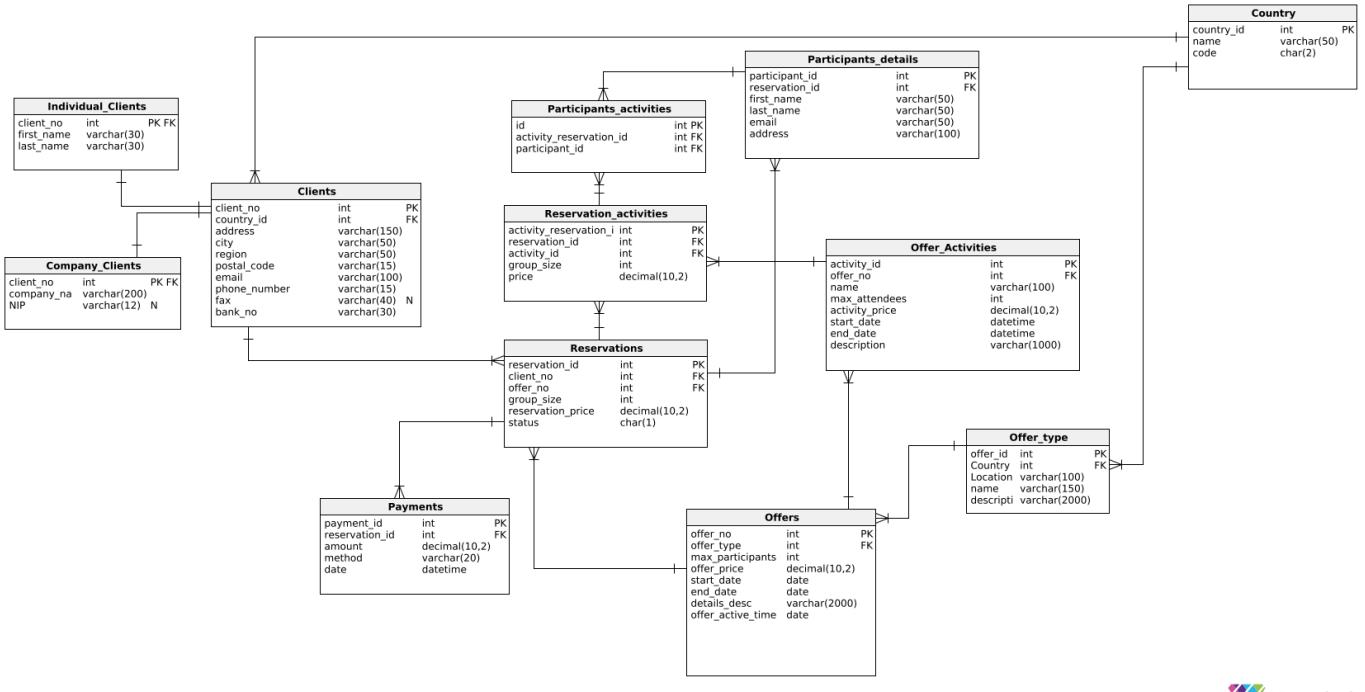
- As an administrator, I want to export data from the database to CSV to generate reports.
- As an employee, I want to be able to add new tour offers and new additional activities to update the company's offerings. - As a user, I want to book a trip for a group of people and be able to select additional activities.
- As a user, I want to pay for the trip in installments to better manage my budget.
- As a user, I want to filter and sort trips by base price, destination, mode of transportation, user ratings, and availability of additional activities I'm interested in, to make it easier to find the best offer.

User Stories v2

- Customer registers the booking with the number of participants
- Adds user data
- Customer selects additional attractions
- Customer makes payment
- Automatic cancellation of the booking by the system
- Customer cancels the booking
- Administrator adds an offer

2. Database

Database schema:



Vertabelo

Description of individual tables:

Clients

Table containing contact and address data for all customers, both individual and corporate.

Attribute Name	Type	Description/Notes
client_no	int	Primary key, customer identifier
country_id	int	Foreign key to Country table
address	varchar(150)	Customer address
city	varchar(50)	City
region	varchar(50)	Region or province
postal_code	varchar(15)	Postal code
email	varchar(100)	Email address
phone_number	varchar(15)	Contact phone number
fax	varchar(40)	Fax number (optional)
bank_no	varchar(50)	Bank account number

- 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

Table storing corporate client data, being a specialization of the Clients table.

Attribute Name	Type	Description/Notes
client_no	int	Primary key, foreign key to Clients
company_name	varchar(200)	Company name
NIP	varchar(12)	NIP number (optional)

- 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	Primary key, foreign key to Clients
first_name	varchar(30)	Customer first name
last_name	varchar(30)	Customer last name

- 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

List of countries available in the system to which clients and offers are assigned.

Attribute Name	Type	Description/Notes
country_id	int	Primary key
name	varchar(50)	Country name
code	char(2)	Two-letter country code

- 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	Primary key
client_no	int	Foreign key to Clients
offer_no	int	Foreign key to Offers
group_size	int	Number of people in the group
reservation_price	decimal(10,2)	Total reservation price
status	char(1)	Reservation status: A - cancelled by employee, K - cancelled by customer, D - pending payment, Z - paid, N - payment deadline exceeded

- 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

Table storing information about payments for reservations.

Attribute Name	Type	Description/Notes
payment_id	int	Primary key
reservation_id	int	Foreign key to Reservations
amount	decimal(10,2)	Payment amount
method	varchar(20)	Payment method (e.g., card, transfer)
date	datetime	Payment date

- 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

Table storing the main offers available to customers.

Attribute Name	Type	Description/Notes
offer_no	int	Primary key
offer_type	int	Foreign key to Offer_type
max_participants	int	Maximum number of participants
offer_price	decimal(10,2)	Base offer price
start_date	date	Start date
end_date	date	End date
details_desc	varchar(2000)	Offer details
offer_active_time	date	Offer activation date

- 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	Primary key
Country	int	Foreign key to Country
Location	varchar(100)	Offer location
name	varchar(150)	Offer type name
description	varchar(2000)	Offer type description

- 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

Table storing information about activities available within offered packages.

Attribute Name	Type	Description/Notes
activity_id	int	Primary key
offer_no	int	Foreign key to Offers
name	varchar(100)	Activity name
max_attendees	int	Maximum number of participants
activity_price	decimal(10,2)	Activity participation price
start_date	datetime	Activity start date
end_date	datetime	Activity end date
description	varchar(1000)	Activity description

- 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

Registers activities assigned to reservations.

Attribute Name	Type	Description
activity_reservation_id	int	Primary key
reservation_id	int	Foreign key to Reservations
activity_id	int	Foreign key to Offer_Activities
group_size	int	Number of activity participants
price	decimal(10,2)	Activity price

- 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

Table containing associations between participants and assigned activities.

Attribute Name	Type	Description
id	int	Primary key
activity_reservation_id	int	Foreign key to Reservation_activities
participant_id	int	Foreign key to 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

Table containing detailed participant data.

Attribute Name	Type	Description
participant_id	int	Primary key
reservation_id	int	Foreign key to Reservations
first_name	varchar(50)	Participant first name
last_name	varchar(50)	Participant last name
email	varchar(50)	Participant email
address	varchar(100)	Participant address

- 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. Views, procedures/functions, triggers

Views

Activity status report – number of reservations, participants, revenues.

```
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;

```

View of available offers with the number of available places and the offer price.

```

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;

```

A view showing the reservation and offer numbers and details of customers whose reservations were canceled due to failure to pay the fee within 7 days before the start of the offer or failure to complete the details of the tour participants within 7 days before the start of the tour.

```

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
    )

```

```
)
);
```

Full information about the customer's order, including details about the trip and activities and their participants (who purchased, who participated, what trip, what activities, dates and unit prices).

```
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;
```

Information on which reservations the people taking part in the activities are assigned to, along with their details and the names of the activities.

```
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;
```

Information about the client's total balance (accountant view).

```
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
    )
) AS amount_due
-
(
    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;

```

Customer balance information (accountant view) for a given reservation.

```

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;

```

List of all reservations with customer details and total price.

```

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;

```

Offer status report – number of reservations, participants, revenues.

```

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;

```

List of offers with assigned activities and their details.

```

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;

```

Participant details and assigned activities.

```

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;

```

A view showing the most frequently used payment methods and what the revenue was from them.

```

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

```

Reservation summary view.

```

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;

```

A subview of the customer balance view, which returns a list of customers and their amount due in the event of an overpayment.

```

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

```

Upcoming activities with details.

```

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();

```

Procedures / functions

Adds a new company client to the database. Inserts data into the `Clients` and `Company_Clients` tables.

```

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

        -- Inserting customer data into the Clients table
        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();

        -- Inserting company data into the Company_Clients table
        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;

```

Adds a new individual client to the database. Inserts data into the `Clients` and `Individual_Clients` tables.

```

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

        -- Inserting customer data into the Clients table
        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();

        -- Inserting individual data into the Individual_Clients table
        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;

```

Adds a new offer to the `Offers` table.

```

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;

    -- Checking the correctness of dates
    IF @EndDate < @StartDate
    BEGIN
        THROW 51113, 'The end date cannot be earlier than the start date.', 1;
    END

    -- Inserting an offer
    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;

```

Adding tour attractions to the `Offer_Activity` table.

```

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;

    -- Checking whether the given offer exists
    IF NOT EXISTS (SELECT 1 FROM Offers WHERE offer_no = @Offer_no)
    BEGIN
        THROW 51113, 'The offer provided does not exist', 1;
    END

    -- Checking the correctness of dates
    IF @end_date < @start_date
    BEGIN
        THROW 51113, 'The end date cannot be earlier than the start date.', 1;
    END

    IF @start_date < (SELECT start_date FROM Offers WHERE offer_no = @Offer_no)
    BEGIN
        THROW 51113, 'the start date must be later than the tour start date', 1;
    END

    IF @end_date > (SELECT end_date FROM Offers WHERE offer_no = @Offer_no)
    BEGIN
        THROW 51113, 'the end date must be earlier than the end date of the trip', 1;
    END

```

```
-- Checking whether the number of places in the trip is less than the maximum
IF @max_group > (SELECT max_participants FROM Offers WHERE offer_no = @Offer_no)
BEGIN
    THROW 51113, 'the maximum number of places must be less than the maximum number of places per trip', 1;
END

-- Inserting an offer
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;
```

Adds a new trip type to the `Offer_type` table.

```
CREATE PROCEDURE dbo.AddOffer_type
    @Country      INT,
    @Location     VARCHAR(100),
    @name         VARCHAR(150),
    @description  VARCHAR(2000)
AS
BEGIN
    SET NOCOUNT ON;
    -- Checking whether a given country exists
    IF NOT EXISTS (SELECT 1 FROM Country where country_id = @Country)
        BEGIN
            THROW 51113, 'There is no such country', 1;
        END

    -- Inserting a trip type
    INSERT INTO dbo.Offer_type
    (Country, Location, name, description)
    VALUES
        (@Country,@Location,@name,@description);

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

Adds a participant association to an activity in the `Participants_activities` table.

```
CREATE PROCEDURE dbo.AddParticipantActivity
    @activity_reservation_id    INT,
    @participant_id             INT
AS
BEGIN TRY

    -- Checking if the specified participant exists
    IF NOT EXISTS (
        SELECT 1
        FROM Participants_details
        WHERE participant_id = @participant_id
    )
    BEGIN
        THROW 50001, 'There is no such participant!', 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
```

```
-- Checking whether the attraction booking is not linked to a booking other than the participant's
IF @participant_res_id <> @activity_res_id
BEGIN
    THROW 50001, 'The attraction booking is linked to a different booking than the participant', 1;
END

DECLARE @reserved INT;

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

-- Checking if all participants have not already been assigned
IF @reserved = (
    SELECT group_size
    FROM Reservation_activities
    WHERE activity_reservation_id = @activity_reservation_id
)
BEGIN
    THROW 50001, 'All participants have already been assigned!', 1;
END

-- Checking whether the participant is not already assigned to a similar activity
IF @participant_id IN (
    SELECT participant_id
    FROM Participants_activities
    WHERE activity_reservation_id = @activity_reservation_id
)
BEGIN
    THROW 50001, 'This participant is already assigned to the specified activity', 1;
END

-- Insertion
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;
```

Adds a participant to the `Participants_details` table associated with the reservation.

```
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

    -- Checking if there is a reservation
    IF NOT EXISTS (
        SELECT 1
        FROM Reservations
        WHERE reservation_id = @Reservation_id
    )
    BEGIN
        THROW 50001, 'You cannot add a participant who is not assigned to the reservation!', 1;
    END

    -- Inserting a participant
    INSERT INTO Participants_details (
        reservation_id, first_name, last_name, email, address
    )
    VALUES (
        @Reservation_id, @First_name, @Last_name, @Email, @Address
    );
END TRY
BEGIN CATCH
    THROW;
END CATCH;
```

```

);
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

        -- Checking if the specified reservation exists
        IF NOT EXISTS (SELECT 1 FROM Reservations WHERE reservation_id=@ReservationID)
        BEGIN
            THROW 50001, 'There is no such reservation', 1;
        END

        -- Adding a payment
        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;

```

Creates a new reservation for the customer for the selected offer.

```

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

        -- Checking if the specified client exists
        IF NOT EXISTS (
            SELECT 1 FROM Clients
            WHERE client_no = @client_no
        )
        BEGIN
            THROW 50001, 'Error - there is no such client', 1;
        END

        -- Checking whether the given offer exists
        IF NOT EXISTS (
            SELECT 1 FROM Offers
            WHERE offer_no = @offer_no
        )
        BEGIN
            THROW 50002, 'Error - there is no such offer', 1;
        END
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;
        THROW;
    END CATCH
END;

```

```

        END

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

-- Insertion
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;

```

Adds an activity to a reservation in the `Reservation_activities` table.

```

CREATE PROCEDURE dbo.AddReservationActivity
    @Reservation_id      INT,
    @Activity_id         INT,
    @group_size          INT
AS
BEGIN
    BEGIN TRANSACTION;
    BEGIN TRY

        -- Checking if the specified reservation exists
        IF NOT EXISTS (
            SELECT 1 FROM Reservations
            WHERE reservation_id = @Reservation_id
        )
        BEGIN
            THROW 50001, 'Error - no such reservation_id', 1;
        END

        -- Checking if the specified activity exists
        IF NOT EXISTS (
            SELECT 1 FROM Offer_Activities
            WHERE activity_id = @Activity_id
        )
        BEGIN
            THROW 50002, 'Error - no such activity_id', 1;
        END
        DECLARE @price DECIMAL(10,2);
        SELECT @price = activity_price FROM Offer_Activities WHERE activity_id = @Activity_id;

        -- Insertion
        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 CATCH
END;
```

Allows an employee to cancel any reservation.

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

        -- Checking whether the specified reservation exists
        IF NOT EXISTS (
            SELECT 1
            FROM Reservations
            WHERE reservation_id = @reservation_id
        )
        BEGIN
            THROW 50002, 'Error - no such reservation', 1;
        END

        -- Reservation status update
        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;
```

Allows the customer to cancel their own reservation.

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

        -- Checking if the customer and reservation exist
        IF NOT EXISTS (
            SELECT 1
            FROM Clients
            WHERE client_no = @client_no
        )
        BEGIN
            THROW 50001, 'Error - there is no such client', 1;
        END

        IF NOT EXISTS (
            SELECT 1
            FROM Reservations
            WHERE reservation_id = @reservation_id
        )
        BEGIN
            THROW 50002, 'Error - no such reservation', 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, 'Error - reservation does not belong to this client', 1;
            END
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;
        THROW;
    END CATCH
END;
```

```

        END

        -- Reservation status update
        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;

```

Sets the booking status to 'A' if all tour participant details are not provided.

```

CREATE PROCEDURE dbo.CancelReservationNoParticipantData
AS
BEGIN

    -- Cancellation of reservation
    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;

```

Updates the reservation status to 'N' if the payment due date has passed.

```

CREATE PROCEDURE dbo.CancelUnpaidReservations
AS
BEGIN

    -- Cancellation of reservation
    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;

```

Allows to change the group size for a given activity.

```

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

        -- Checking if there is such an activity reservation
        IF NOT EXISTS (
            SELECT 1 FROM Reservation_activities WHERE activity_reservation_id = @activity_reservation_id
        )

```

```

BEGIN
    THROW 50002, 'Error - no such activity reservation exists', 1;
END

-- Checking the number of participants (if there are too many)
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, 'Error - too many participants are assigned to the activity', 1;
END

-- Changing group size
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;

```

Allows you to edit tour participant details.

```

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

        -- Checking if such a participant exists
        IF NOT EXISTS (
            SELECT 1 FROM Participants_details WHERE participant_id = @participant_id
        )
        BEGIN
            THROW 50002, 'Error - no such participant exists', 1;
        END

        -- Editing tour participant data
        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;

```

Allows you to change the participant or activity in the association.

```

CREATE PROCEDURE dbo.EditParticipantActivity
    @id                      INT,
    @participant_id           INT,
    @new_reservation_activity INT
AS
BEGIN
    BEGIN TRANSACTION;
    BEGIN TRY

        -- Checking whether a given participant assignment to a trip exists
        IF NOT EXISTS (
            SELECT 1 FROM Participants_activities
            WHERE id = @id
        )
        BEGIN
            THROW 50002, 'Error - there is no such participant assignment to the trip', 1;
        END

        -- Checking if the specified participant exists
        IF NOT EXISTS (
            SELECT 1 FROM Participants_details
            WHERE participant_id = @participant_id
        )
        BEGIN
            THROW 50002, 'Error - no such participant exists', 1;
        END

        -- Checking if the specified reservation exists
        IF NOT EXISTS (
            SELECT 1 FROM Reservation_activities
            WHERE activity_reservation_id = @new_reservation_activity
        )
        BEGIN
            THROW 50002, 'Error - no such activity reservation exists', 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;

        -- Checking whether participants do not belong to the same trips
        IF @old_participant_res_id <> @new_participant_res_id
        BEGIN
            THROW 50002, 'Error - participants belong to different trips', 1;
        END

        -- Change
        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;

```

Allows you to edit the order (change the number of participants).

```

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

        -- Checking if the specified reservation exists
        IF NOT EXISTS (
            SELECT 1 FROM Reservations r
            WHERE r.reservation_id = @reservation_id
        )
        BEGIN
            THROW 50002, 'Error - no such reservation exists', 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, 'Error - the number of participants associated with the reservation is greater than the new group size', 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, 'Error - one of the activities is larger than the new group size', 1;
        END

        -- Change in the number of participants
        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;

```

Deletes the activity booking and associated participant assignments.

```

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

        -- Checking if there is such an activity reservation
        IF NOT EXISTS (
            SELECT 1 FROM Reservation_activities WHERE activity_reservation_id = @activity_reservation_id
        )
        BEGIN
            THROW 50002, 'Error - no such activity reservation exists', 1;
        END
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;
        THROW;
    END CATCH
END;

```

```
-- Deleting an activity reservation
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;
```

Deletes the participant and all activity links.

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

        -- Checking whether the specified activity participant exists
        IF NOT EXISTS (
            SELECT 1 FROM Participants_details pd
            WHERE pd.participant_id = @participant_id
        )
        BEGIN
            THROW 50002, 'Error - the specified participant does not exist', 1;
        END

        -- Removing a participant
        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;
```

Removes the participant's association with an activity.

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

        -- Checking whether a participant was assigned to a given activity
        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, 'Error - participant is not assigned to the given activity', 1;
        END

        -- Removing a participant's association with an activity
        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;

```

History of all activities in which a given customer participated (purchased).

```

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
);

```

The `fn_ClientReservations` function generates a list of all reservations made by the client with the given `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
);

```

Returns summary information about customer payments – total amount, number of payments, average payment.

```

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
);

```

Triggers

Trigger automatically sets the booking status to "Z" if the total payments cover the cost of the booking and the activity.

```

CREATE TRIGGER increase_status_based_on_amount
ON Payments
AFTER INSERT
AS
BEGIN

    -- Receiving the reservation price
    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;

    -- Download the activity price
    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;

    -- Updating the reservation status to "Z" if the total payments >= reservation cost + activities
    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;

```

Checking whether there are enough places for the group on the trip and whether it is still possible to edit the order - activated when adding a new reservation and when editing an existing reservation (our database does not allow for deleting records from the reservations table, only for changing the status).

```

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;

    -- Checking if reservations can be edited
    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, 'Error - booking can no longer be edited', 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, 'Error - group size is too large', 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, 'Error - new group size is too large', 1;
                    ROLLBACK TRANSACTION;
                    RETURN;
                END
        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;

```

Checking whether it is possible to edit an activity reservation - checks whether the deadline for changes has not passed and whether there are enough places in the case of adding a new reservation or editing an existing reservation.

```

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;

```

```
-- Checking if reservations can be edited
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, 'Error - booking can no longer be edited', 1;
    END

    -- Checking the number of people
    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, 'Error - the number of people exceeds the number registered for the trip', 1;
    END

    -- Checking availability of places
    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, 'No activity spaces available', 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, 'No places available for new group', 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, 'Error - reservation can no longer be deleted', 1;
END

DELETE Reservation_activities WHERE reservation_id = @reservation_id;
END
END;

```

Trigger checks whether it is still possible to edit participant data (add, delete, change).

```

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

    -- Get the tour start date
    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

    -- Checking whether participant data can still be edited
    IF DATEDIFF(DAY, GETDATE(), @start_date) <= 7
    BEGIN
        THROW 50002, 'Error - participant details for this trip can no longer be edited', 1;
    END
END;

```

Trigger checks whether it is still possible to edit participant assignments to attractions.

```

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

    -- Getting the tour start date
    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;

-- Checking if you can still edit attraction assignments
IF DATEDIFF(DAY, GETDATE(), @start_date) <= 7
BEGIN
    THROW 50002, 'Error - You can no longer edit attraction assignments for participants on this trip', 1;
END
END;

```

4. Creating roles:

Seller and customer contact person

This role is intended for customer-facing employees, including salespeople and reservations managers. Users assigned to this role can view, add, and edit customer data, reservations, attendees, and payments. They also have access to basic information about the offer and its associated activities.

```
CREATE ROLE Customer_Service;
```

```

-- Permissions to Tables
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;

-- Permissions to Views
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;

-- Authorization to Procedures
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, whose task is to manage offers and activities.

This role is dedicated to managers responsible for creating and managing offers and related activities. Users in this role can view, add, modify, and delete data related to offers, offer types, and activities.

```
CREATE ROLE Offer_Manager;
```

```
-- Permissions to Tables
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;

-- Permissions to Views
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;

-- Authorization to Procedures
GRANT EXECUTE ON AddOffer TO Offer_Manager;
GRANT EXECUTE ON AddOffer_type TO Offer_Manager;
GRANT EXECUTE ON AddOffer_Activity TO Offer_Manager;
```

Accountant

This role is designed for the accounting department. Users assigned to this role have only read access to data required for financial settlements, including customer information, reservations, payments, and related activities.

```
CREATE ROLE Accountant;
```

```
-- Permissions to Tables
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;

-- Permissions to Views
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;
```

Database Administrator

The Administrator user with SELECT, VIEW ANY DATABASE, BACKUP DATABASE and ADMINISTER BULK OPERATIONS privileges can view data in the databases to which he or she has access, see a list of all databases on the server, perform backups and use bulk data import and export operations.

```
CREATE ROLE Administrator;
```

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

Database Client

Client permissions are defined at the application layer. Each client receives an individual account, with a login consisting of a combination of first name, last name, and a string of numbers. The password is automatically generated and is 12 characters long.

5. Indexes

Index on (reservation_id, date) in the Payments table

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

Index on (client_no, offer_no) in the Reservations table

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

Index on (reservation_id, activity_id) in the Reservation_activities table

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

Index on (activity_reservation_id, participant_id) in the Participants_activities table

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

Index on (reservation_id) in the Participants_details table

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

Index on (offer_no) in the Offer_Activities table

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

Index on (offer_type, start_date, offer_active_time) in the Offers table

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

Index to (Country) in table Offer_type

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