

Sprawozdanie Bazy Danych

Mieszko Makowski, Karol Mierzwiński, Mateusz Wirkijowski, Patryk Motylski

Agenda:

- funkcje systemu
- schemat
- warunki integralności
- szczegółowy opis każdej tabeli
 - nazwa tabeli
 - nazwa atrybutu
 - warunki integralności (PK, FK, default, unique, check)
- widoki/procedury
- dane
- funkcje/triggery
- uprawnienia

Uprawnienia do funkcji systemu:

(wszyscy muszą być zalogowani oprócz potencjalnego klienta)

Rola	Funkcje	Widok
Administrator	9. Generowanie raportów 15. Zarządzanie wycieczkami 16. Zarządzanie aktywnościami dodatkowymi	1.Cena za dane zamówienie 7.Najwyższe zapotrzebowanie na wycieczki
Sprzedawca/ Reklamodawca	1. Wykonywanie rezerwacji wycieczki 5. Wyświetlanie wszystkich wycieczek 10. Opinie klienckie	1.Cena za dane zamówienie 3.Najczęściej wybierane wycieczki 4.Najczęściej wybierane atrakcje 5.Najczęściej wybierane serwisy 9.Historia zamówień klienta 10.Dostępne wolne miejsca na wycieczce
Kierownik wycieczki	3. Podanie osób biorących udział w wycieczce 4. Rezerwacja dodatkowych usług i atrakcji do danej wycieczki 6. Modyfikacja danej wycieczki, która została zarezerwowana/opłacona 8. Anulowanie rezerwacji/atrakcji/dodatkowych usług 10. Opinie klienckie 12. Transport <ul style="list-style-type: none"> przekazywanie do Administratora modyfikacji wycieczek (np. zmiana ilości uczestników) 	2.Uczestnicy danej wycieczki 8.Nieopłacone zamówienia na tydzień przed
Przewodnik wycieczki	3. Podanie osób biorących udział w wycieczce	2.Uczestnicy wycieczki
Prowadzący aktywność dodatkową	14. Podanie osób biorących udział w aktywności dodatkowej	
Uczestnik	1. Wykonywanie rezerwacji wycieczki 2. Dokonanie opłaty za wycieczkę 4. Rezerwacja dodatkowych usług i atrakcji do danej wycieczki 5. Wyświetlanie wszystkich wycieczek 7. Potwierdzenie rezerwacji 11. Pobieranie Faktur i paragonów z dokonanych transakcji	1.Cena za dane zamówienie 3.Najczęściej wybierane wycieczki 4.Najczęściej wybierane atrakcje 5.Najczęściej wybierane serwisy 6.Zamówienia oczekujące płatności 9.Historia zamówień klienta 10.Dostępne wolne miejsca na wycieczce
Uczestnik (firma)	1. Wykonywanie rezerwacji wycieczki 2. Dokonanie opłaty za wycieczkę 4. Rezerwacja dodatkowych usług i atrakcji do danej wycieczki 5. Wyświetlanie wszystkich wycieczek 7. Potwierdzenie rezerwacji 11. Pobieranie Faktur i paragonów z dokonanych transakcji	1.Cena za dane zamówienie 3.Najczęściej wybierane wycieczki 4.Najczęściej wybierane atrakcje 5.Najczęściej wybierane serwisy 6.Zamówienia oczekujące płatności 9.Historia zamówień klienta 10.Dostępne wolne miejsca na wycieczce

Potencjalny Klient (niezalogowany)	1. Wykonywanie rezerwacji wycieczki 3. Podanie osób biorących udział w wycieczce 5. Wyświetlanie wszystkich wycieczek	1.Cena za dane zamówienie 3.Najczęściej wybierane wycieczki 4.Najczęściej wybierane atrakcje 5.Najczęściej wybierane serwisy 10.Dostępne wolne miejsca na wycieczce
---	---	---

FUNKCJE :

1. Wykonywanie rezerwacji wycieczki

SQL Potrzebne: INSERT do tabeli rezerwacji.

Uprawnienia do dodawania rekordów w tabeli rezerwacji.
2. Dokonanie opłaty za wycieczkę

SQL Potrzebne: UPDATE na tabeli rezerwacji, INSERT do tabeli płatności.

Uprawnienia do aktualizacji rekordu rezerwacji i dodawania rekordów do tabeli płatności.
3. Podanie osób biorących udział w wycieczce

SQL Potrzebne: INSERT do tabeli uczestników.

Uprawnienia do dodawania rekordów w tabeli uczestników.
4. Rezerwacja dodatkowych usług i atrakcji do danej wycieczki

SQL Potrzebne: INSERT do tabeli dodatkowych usług.

Uprawnienia do dodawania rekordów w tabeli dodatkowych usług.
5. Wyświetlanie wszystkich wycieczek

SQL Potrzebne: SELECT na tabeli wycieczek.

Uprawnienia do czytania danych z tabeli wycieczek.
6. Modyfikacja danej wycieczki, która została zarezerwowana/opłacona

SQL Potrzebne: UPDATE na tabeli wycieczek i rezerwacji.

Uprawnienia do aktualizacji danych w tabeli wycieczek i rezerwacji.
7. Potwierdzenie rezerwacji

SQL Potrzebne: UPDATE na tabeli rezerwacji.

Uprawnienia do aktualizacji rekordu rezerwacji.
8. Anulowanie rezerwacji/atrakcji/dodatkowych usług

SQL Potrzebne: DELETE lub UPDATE (ustawienie statusu na anulowany).

Uprawnienia do usuwania lub aktualizacji rekordów w odpowiednich tabelach.

9. Generowanie raportów

SQL Potrzebne: SELECT na wielu tabelach.

Uprawnienia do czytania z wielu tabel.

10. Opinie klienckie

SQL Potrzebne: INSERT, SELECT, UPDATE na tabeli opinii.

Uprawnienia do dodawania, aktualizowania i czytania z tabeli opinii.

11. Pobieranie Faktur i paragonów z dokonanych transakcji

SQL Potrzebne: SELECT na tabeli transakcji i faktur.

Uprawnienia do czytania z tych tabel.

12. Transport

SQL Potrzebne: INSERT, UPDATE, SELECT na tabeli transportu.

Uprawnienia do dodawania, aktualizowania i czytania z tabeli transportu.

13. Zarządzanie dokumentami klientów

SQL Potrzebne: INSERT, UPDATE, SELECT na tabeli dokumentów.

Uprawnienia do zarządzania dokumentami

14. Podanie osób biorących udział w aktywności dodatkowej

SQL Potrzebne: INSERT do tabeli uczestników.

Uprawnienia do dodawania rekordów w tabeli uczestników.

15. Zarządzanie wycieczkami

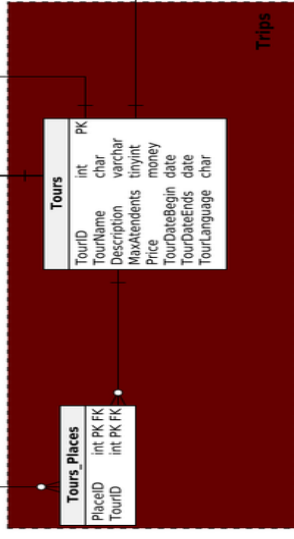
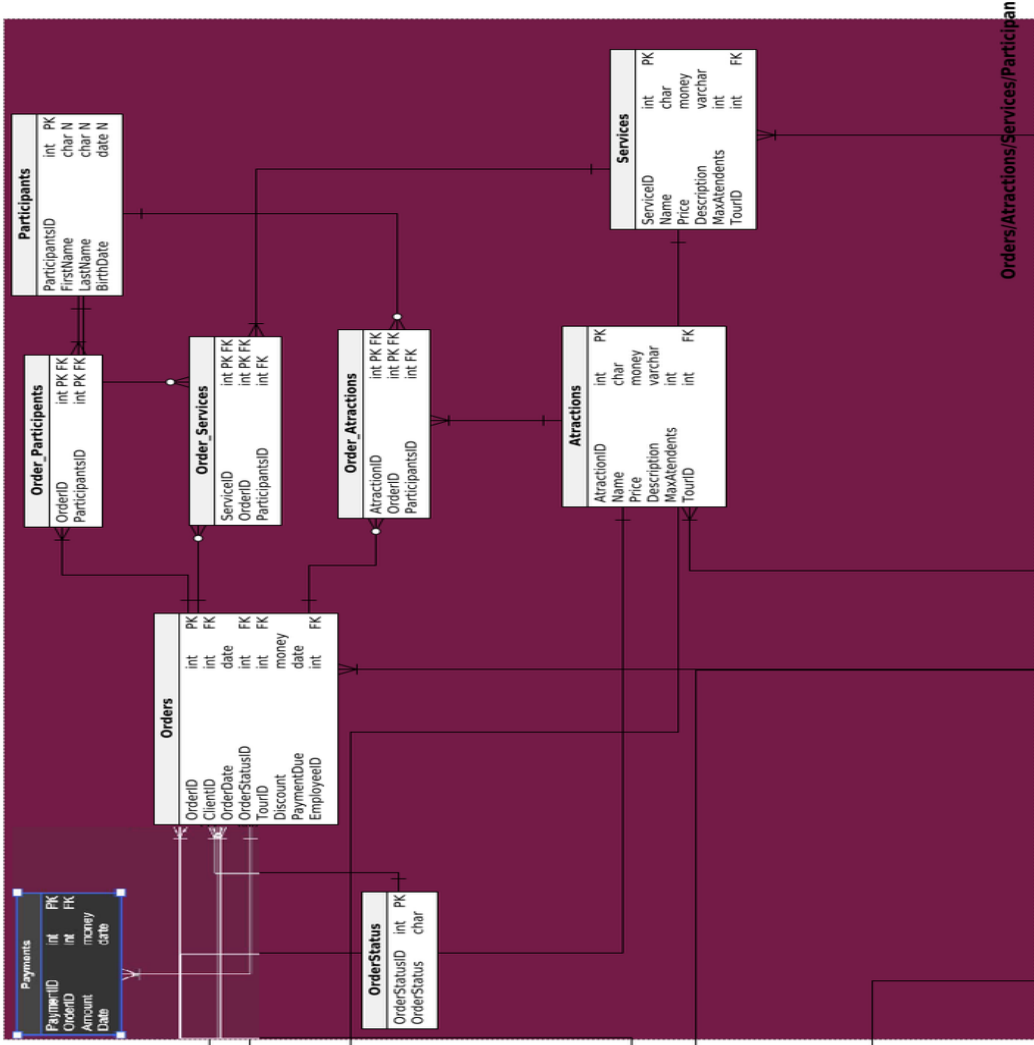
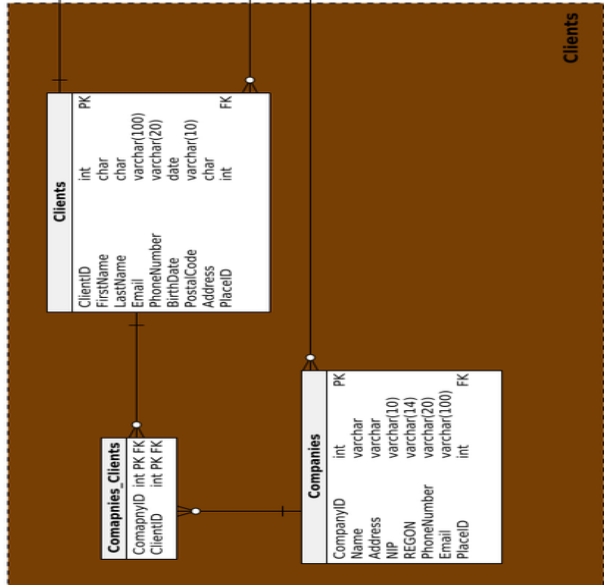
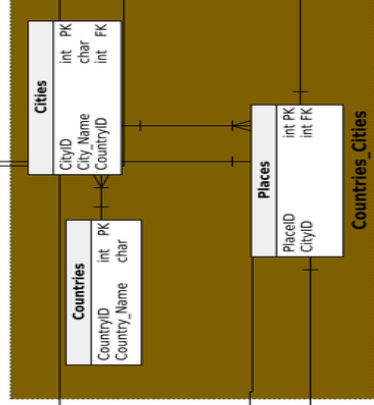
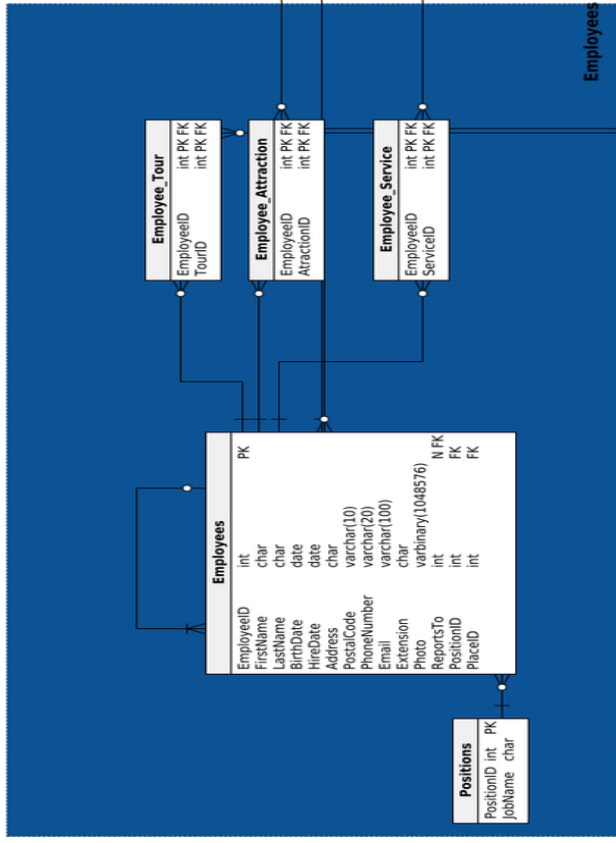
SQL Potrzebne: INSERT, UPDATE, SELECT,DELETE na tabeli wycieczek.

16. Zarządzanie aktywnościami dodatkowymi

SQL Potrzebne: INSERT, UPDATE, SELECT,DELETE na tabeli aktywności.

Schemat:

[Link do schematu](#)



Opis tabel:

Table: Atractions

Description: Przechowuje informacje o atrakcjach turystycznych

- **AtractionID** - Unikalny identyfikator atrakcji, typ danych: int
- **Name** - Nazwa atrakcji, typ danych: char
- **Price** - Cena atrakcji, typ danych: money
- **Description** - Opis atrakcji, typ danych: varchar
- **MaxAtendents** - Maksymalna liczba uczestników, typ danych: int
- **TourID** - Identyfikator wycieczki, typ danych: int (FK)
- **Atractions_ak_1** - Warunek integralności: unikalna nazwa atrakcji
- **Atractions_pk** - Warunek integralności: klucz główny

SQL Code:

```
CREATE TABLE Atractions (  
  AtractionID int NOT NULL,  
  Name char NOT NULL,  
  Price money NOT NULL,  
  Description varchar NOT NULL,  
  MaxAtendents int NOT NULL,  
  TourID int NOT NULL,  
  CONSTRAINT Atractions_ak_1 UNIQUE (Name),  
  CONSTRAINT Atractions_pk PRIMARY KEY (AtractionID)  
);
```

Table: Cities

Description: Przechowuje informacje o miastach

- **CityID** - Unikalny identyfikator miasta, typ danych: int
- **City_Name** - Nazwa miasta, typ danych: char
- **CountryID** - Identyfikator kraju, typ danych: int (FK)
- **Cities_pk** - Warunek integralności: klucz główny

SQL Code:

```
CREATE TABLE Cities (  
  CityID int NOT NULL,  
  City_Name char NOT NULL,  
  CountryID int NOT NULL,  
  CONSTRAINT Cities_pk PRIMARY KEY (CityID)  
);
```

Table: Clients

Description: Przechowuje informacje o klientach

- **ClientID** - Unikalny identyfikator klienta, typ danych: int
- **FirstName** - Imię klienta, typ danych: char
- **LastName** - Nazwisko klienta, typ danych: char
- **Email** - Email klienta, typ danych: varchar(100)
- **PhoneNumber** - Numer telefonu klienta, typ danych: varchar(20)
- **BirthDate** - Data urodzenia klienta, typ danych: date
- **PostalCode** - Kod pocztowy klienta, typ danych: varchar(10)
- **Address** - Adres klienta, typ danych: char
- **PlaceID** - Identyfikator miejsca, typ danych: int (FK)
- **Clients_ak_1** - Warunek integralności: unikalność email i numeru telefonu
- **Email** - Warunek integralności: sprawdzenie poprawności email
- **PhoneNumber** - Warunek integralności: sprawdzenie poprawności numeru telefonu
- **Address** - Warunek integralności: sprawdzenie poprawności adresu
- **Clients_pk** - Warunek integralności: klucz główny

SQL Code:

```
CREATE TABLE Clients (  
  ClientID int NOT NULL,  
  FirstName char NOT NULL,  
  LastName char NOT NULL,  
  Email varchar(100) NOT NULL,  
  PhoneNumber varchar(20) NOT NULL,  
  BirthDate date NOT NULL,  
  PostalCode varchar(10) NOT NULL,  
  Address char NOT NULL,  
  PlaceID int NOT NULL,  
  CONSTRAINT Clients_ak_1 UNIQUE (Email, PhoneNumber),  
  CONSTRAINT Email CHECK ([w-\.]+@[w-\.]+\w]{2,4}$),  
  CONSTRAINT PhoneNumber CHECK ([+]?([0-9]{3})?[-\s\.]?[0-9]{3}[-\s\.]?[0-9]{4,6}$),  
  CONSTRAINT Address CHECK ([d{1,5}\s\w\s(b\w*\b\s){1,2}\w*\.),  
  CONSTRAINT Clients_pk PRIMARY KEY (ClientID)  
);
```

Table: Comapnies

Description: Przechowuje informacje o firmach

- **CompanyID** - Unikalny identyfikator firmy, typ danych: int
- **Name** - Nazwa firmy, typ danych: varchar
- **Address** - Adres firmy, typ danych: varchar
- **NIP** - Numer NIP firmy, typ danych: varchar(10)
- **REGON** - Numer REGON firmy, typ danych: varchar(14)
- **PhoneNumber** - Numer telefonu firmy, typ danych: varchar(20)

- **Email** - Email firmy, typ danych: varchar(100)
- **PlaceID** - Identyfikator miejsca, typ danych: int (FK)
- **Firms_ak_1** - Warunek integralności: unikalność NIP, REGON, numeru telefonu i emaila
- **Address** - Warunek integralności: sprawdzenie poprawności adresu
- **NIP** - Warunek integralności: sprawdzenie poprawności NIP
- **PhoneNumber** - Warunek integralności: sprawdzenie poprawności numeru telefonu
- **Email** - Warunek integralności: sprawdzenie poprawności email
- **Comapanies_pk** - Warunek integralności: klucz główny

SQL Code:

```
CREATE TABLE Comapnies (
  CompanyID int NOT NULL,
  Name varchar NOT NULL,
  Address varchar NOT NULL,
  NIP varchar(10) NOT NULL,
  REGON varchar(14) NOT NULL,
  PhoneNumber varchar(20) NOT NULL,
  Email varchar(100) NOT NULL,
  PlaceID int NOT NULL,
  CONSTRAINT Firms_ak_1 UNIQUE (NIP, REGON, PhoneNumber, Email),
  CONSTRAINT Address CHECK (\d{1,5}\s\w.\s(\b\w*\b\s){1,2}\w*\.),
  CONSTRAINT NIP CHECK (- ^((\d{3}[- ]\d{3}[- ]\d{2}[- ]\d{2})|(\d{3}[- ]\d{2}[- ]\d{2}[- ]\d{3}))$),
  CONSTRAINT PhoneNumber CHECK (^[\+]?([]?[0-9]{3}[I])?[-\s\.]?[0-9]{3}[-\s\.]?[0-9]{4,6}$),
  CONSTRAINT Email CHECK (^[\w-\.] + @([\w-]+ \.) + [\w-]{2,4}$),
  CONSTRAINT Comapanies_pk PRIMARY KEY (CompanyID)
);
```

Table: Comapnies_Clients

Description: Przechowuje informacje o relacjach między firmami a klientami

- **ComapnyID** - Identyfikator firmy, typ danych: int (FK)
- **ClientID** - Identyfikator klienta, typ danych: int (FK)
- **Companies Clients pk** - Warunek integralności: klucz główny złożony

SQL Code:

```
CREATE TABLE Comapnies_Clients (
  ComapnyID int NOT NULL,
  ClientID int NOT NULL,
  CONSTRAINT Comapnies_Clients_pk PRIMARY KEY (ComapnyID, ClientID)
);
```

Table: Countries

Description: Przechowuje informacje o krajach

- **CountryID** - Unikalny identyfikator kraju, typ danych: int
- **Country_Name** - Nazwa kraju, typ danych: char
- **Countries_ak_1** - Warunek integralności: unikalność nazwy kraju
- **Countries_pk** - Warunek integralności: klucz główny

SQL Code:

```
CREATE TABLE Countries (  
  CountryID int NOT NULL,  
  Country_Name char NOT NULL,  
  CONSTRAINT Countries_ak_1 UNIQUE (Country_Name),  
  CONSTRAINT Countries_pk PRIMARY KEY (CountryID)  
);
```

Table: Employee_Attraction

Description: Przechowuje informacje o relacjach między pracownikami a atrakcjami

- **EmployeeID** - Identyfikator pracownika, typ danych: int (FK)
- **AtractionID** - Identyfikator atrakcji, typ danych: int (FK)
- **Employee_Attraction_pk** - Warunek integralności: klucz główny złożony

SQL Code:

```
CREATE TABLE Employee_Attraction (  
  EmployeeID int NOT NULL,  
  AtractionID int NOT NULL,  
  CONSTRAINT Employee_Attraction_pk PRIMARY KEY (EmployeeID, AtractionID)  
);
```

Table: Employee_Service

Description: Przechowuje informacje o relacjach między pracownikami a usługami

- **EmployeeID** - Identyfikator pracownika, typ danych: int (FK)
- **ServiceID** - Identyfikator usługi, typ danych: int (FK)
- **Employee_Service_pk** - Warunek integralności: klucz główny złożony

SQL Code:

```
CREATE TABLE Employee_Service (  
  EmployeeID int NOT NULL,  
  ServiceID int NOT NULL,  
  CONSTRAINT Employee_Service_pk PRIMARY KEY (EmployeeID, ServiceID)  
);
```

Table: Employee_Tour

Description: Przechowuje informacje o relacjach między pracownikami a wycieczkami

- **EmployeeID** - Identyfikator pracownika, typ danych: int (FK)
- **TourID** - Identyfikator wycieczki, typ danych: int (FK)
- **Employee_Tour_pk** - Warunek integralności: klucz główny złożony

SQL Code:

```
CREATE TABLE Employee_Tour (  
    EmployeeID int NOT NULL,  
    TourID int NOT NULL,  
    CONSTRAINT Employee_Tour_pk PRIMARY KEY (EmployeeID, TourID)  
);
```

Table: Employees

Description: Przechowuje informacje o pracownikach

- **EmployeeID** - Unikalny identyfikator pracownika, typ danych: int
- **FirstName** - Imię pracownika, typ danych: char
- **LastName** - Nazwisko pracownika, typ danych: char
- **BirthDate** - Data urodzenia pracownika, typ danych: date
- **HireDate** - Data zatrudnienia pracownika, typ danych: date
- **Address** - Adres pracownika, typ danych: char
- **PostalCode** - Kod pocztowy pracownika, typ danych: varchar(10)
- **PhoneNumber** - Numer telefonu pracownika, typ danych: varchar(20)
- **Email** - Email pracownika, typ danych: varchar(100)
- **PositionID** - Identyfikator stanowiska, typ danych: int (FK)
- **PlaceID** - Identyfikator miejsca, typ danych: int (FK)
- **ReportsTo** - Identyfikator przełożonego, typ danych: int (FK, nullable)
- **Employees_ak_1** - Warunek integralności: unikalność numeru telefonu i emaila
- **Email** - Warunek integralności: sprawdzenie poprawności email
- **PhoneNumber** - Warunek integralności: sprawdzenie poprawności numeru telefonu
- **Address** - Warunek integralności: sprawdzenie poprawności adresu
- **Employees_pk** - Warunek integralności: klucz główny

SQL Code:

```
CREATE TABLE Employees (  
    EmployeeID int NOT NULL,  
    FirstName char NOT NULL,  
    LastName char NOT NULL,  
    BirthDate date NOT NULL,  
    HireDate date NOT NULL,  
    Address char NOT NULL,  
    PostalCode varchar(10) NOT NULL,  
    PhoneNumber varchar(20) NOT NULL,
```

```

Email varchar(100) NOT NULL,
PositionID int NOT NULL,
PlaceID int NOT NULL,
ReportsTo int NULL,
CONSTRAINT Employees_ak_1 UNIQUE (PhoneNumber, Email),
CONSTRAINT Email CHECK ([w-\.] + @([w-]+\.)+[w-]{2,4}$),
CONSTRAINT PhoneNumber CHECK ([\+] ? ([0-9]{3}) ? [-\s\.] ? [0-9]{3} [-\s\.] ? [0-9]{4,6}$),
CONSTRAINT Address CHECK (\d{1,5} \s w.\s (\b w* \b s) {1,2} w* \.),
CONSTRAINT Employees_pk PRIMARY KEY (EmployeeID)
);

```

Table: Order_Atractions

Description: Przechowuje informacje o atrakcjach w zamówieniach

- **OrderID** - Identyfikator zamówienia, typ danych: int (FK)
- **AttractionID** - Identyfikator atrakcji, typ danych: int (FK)
- **ParticipantsID** - Identyfikator uczestników, typ danych: int (FK)
- **Order_Atractions_pk** - Warunek integralności: klucz główny złożony

SQL Code:

```

CREATE TABLE Order_Atractions (
    OrderID int NOT NULL,
    AttractionID int NOT NULL,
    ParticipantsID int NOT NULL,
    CONSTRAINT Order_Atractions_pk PRIMARY KEY (OrderID, AttractionID,
    ParticipantsID)
);

```

Table: Order_Participants

Description: Przechowuje informacje o uczestnikach w zamówieniach

- **OrderID** - Identyfikator zamówienia, typ danych: int (FK)
- **ParticipantsID** - Identyfikator uczestników, typ danych: int (FK)
- **Order_Participants_pk** - Warunek integralności: klucz główny złożony

SQL Code:

```

CREATE TABLE Order_Participants (
    OrderID int NOT NULL,
    ParticipantsID int NOT NULL,
    CONSTRAINT Order_Participants_pk PRIMARY KEY (OrderID, ParticipantsID)
);

```

Table: Order_Services

Description: Przechowuje informacje o usługach w zamówieniach

- **OrderID** - Identyfikator zamówienia, typ danych: int (FK)
- **ServiceID** - Identyfikator usługi, typ danych: int (FK)
- **ParticipantsID** - Identyfikator uczestników, typ danych: int (FK)
- **Order_Services_pk** - Warunek integralności: klucz główny złożony

SQL Code:

```
CREATE TABLE Order_Services (  
  OrderID int NOT NULL,  
  ServiceID int NOT NULL,  
  ParticipantsID int NOT NULL,  
  CONSTRAINT Order_Services_pk PRIMARY KEY (OrderID, ServiceID, ParticipantsID)  
);
```

Table: OrderStatus

Description: Przechowuje informacje o statusach zamówień

- **OrderStatusID** - Unikalny identyfikator statusu zamówienia, typ danych: int
- **Status** - Nazwa statusu, typ danych: char
- **OrderStatus_ak_1** - Warunek integralności: unikalność nazwy statusu
- **OrderStatus_pk** - Warunek integralności: klucz główny

SQL Code:

```
CREATE TABLE OrderStatus (  
  OrderStatusID int NOT NULL,  
  Status char NOT NULL,  
  CONSTRAINT OrderStatus_ak_1 UNIQUE (Status),  
  CONSTRAINT OrderStatus_pk PRIMARY KEY (OrderStatusID)  
);
```

Table: Orders

Description: Przechowuje informacje o zamówieniach

- **OrderID** - Unikalny identyfikator zamówienia, typ danych: int
- **ClientID** - Identyfikator klienta, typ danych: int (FK)
- **EmployeeID** - Identyfikator pracownika, typ danych: int (FK)
- **OrderStatusID** - Identyfikator statusu zamówienia, typ danych: int (FK)
- **TourID** - Identyfikator wycieczki, typ danych: int (FK)
- **OrderDate** - Data zamówienia, typ danych: datetime
- **TotalAmount** - Całkowita kwota zamówienia, typ danych: money
- **Orders_pk** - Warunek integralności: klucz główny

SQL Code:

```
CREATE TABLE Orders (  
  OrderID int NOT NULL,  
  ClientID int NOT NULL,  
  EmployeeID int NOT NULL,  
  OrderStatusID int NOT NULL,  
  TourID int NOT NULL,  
  OrderDate datetime NOT NULL,  
  TotalAmount money NOT NULL,  
  CONSTRAINT Orders_pk PRIMARY KEY (OrderID)  
);
```

Table: Participants

Description: Przechowuje informacje o uczestnikach

- **ParticipantsID** - Unikalny identyfikator uczestnika, typ danych: int
- **FirstName** - Imię uczestnika, typ danych: char
- **LastName** - Nazwisko uczestnika, typ danych: char
- **BirthDate** - Data urodzenia uczestnika, typ danych: date
- **Address** - Adres uczestnika, typ danych: char
- **PostalCode** - Kod pocztowy uczestnika, typ danych: varchar(10)
- **PhoneNumber** - Numer telefonu uczestnika, typ danych: varchar(20)
- **Email** - Email uczestnika, typ danych: varchar(100)
- **PlaceID** - Identyfikator miejsca, typ danych: int (FK)
- **Participants_ak_1** - Warunek integralności: unikalność numeru telefonu i emaila
- **Email** - Warunek integralności: sprawdzenie poprawności email
- **PhoneNumber** - Warunek integralności: sprawdzenie poprawności numeru telefonu
- **Address** - Warunek integralności: sprawdzenie poprawności adresu
- **Participants_pk** - Warunek integralności: klucz główny

SQL Code:

```
CREATE TABLE Participants (  
  ParticipantsID int NOT NULL,  
  FirstName char NOT NULL,  
  LastName char NOT NULL,  
  BirthDate date NOT NULL,  
  Address char NOT NULL,  
  PostalCode varchar(10) NOT NULL,  
  PhoneNumber varchar(20) NOT NULL,  
  Email varchar(100) NOT NULL,  
  PlaceID int NOT NULL,  
  CONSTRAINT Participants_ak_1 UNIQUE (PhoneNumber, Email),  
  CONSTRAINT Email CHECK ([w-\.]+@[w-+\.]+\w-]{2,4}$),  
  CONSTRAINT PhoneNumber CHECK ([+]?([0-9]{3})?[-\s\.]?[0-9]{3}[-\s\.]?[0-9]{4,6}$),  
  CONSTRAINT Address CHECK ([d{1,5}\s\w.\s(b\w*\b\s){1,2}\w*\.),
```

```
CONSTRAINT Participants_pk PRIMARY KEY (ParticipantsID)
);
```

Table: Places

Description: Przechowuje informacje o miejscach

- **PlaceID** - Unikalny identyfikator miejsca, typ danych: int
- **Place_Name** - Nazwa miejsca, typ danych: char
- **CityID** - Identyfikator miasta, typ danych: int (FK)
- **Places_ak_1** - Warunek integralności: unikalność nazwy miejsca
- **Places_pk** - Warunek integralności: klucz główny

SQL Code:

```
CREATE TABLE Places (
  PlaceID int NOT NULL,
  Place_Name char NOT NULL,
  CityID int NOT NULL,
  CONSTRAINT Places_ak_1 UNIQUE (Place_Name),
  CONSTRAINT Places_pk PRIMARY KEY (PlaceID)
);
```

Table: Positions

Description: Przechowuje informacje o stanowiskach

- **PositionID** - Unikalny identyfikator stanowiska, typ danych: int
- **Position** - Nazwa stanowiska, typ danych: char
- **Positions_ak_1** - Warunek integralności: unikalność nazwy stanowiska
- **Positions_pk** - Warunek integralności: klucz główny

SQL Code:

```
CREATE TABLE Positions (
  PositionID int NOT NULL,
  Position char NOT NULL,
  CONSTRAINT Positions_ak_1 UNIQUE (Position),
  CONSTRAINT Positions_pk PRIMARY KEY (PositionID)
);
```

Table: Services

Description: Przechowuje informacje o usługach

- **ServiceID** - Unikalny identyfikator usługi, typ danych: int
- **Name** - Nazwa usługi, typ danych: char
- **Description** - Opis usługi, typ danych: varchar
- **Price** - Cena usługi, typ danych: money

- **Duration** - Czas trwania usługi w minutach, typ danych: int
- **TourID** - Identyfikator wycieczki, typ danych: int (FK)
- **Services_ak_1** - Warunek integralności: unikalność nazwy usługi
- **Services_pk** - Warunek integralności: klucz główny

SQL Code:

```
CREATE TABLE Services (
  ServiceID int NOT NULL,
  Name char NOT NULL,
  Description varchar NOT NULL,
  Price money NOT NULL,
  Duration int NOT NULL,
  TourID int NOT NULL,
  CONSTRAINT Services_ak_1 UNIQUE (Name),
  CONSTRAINT Services_pk PRIMARY KEY (ServiceID)
);
```

Table: Tours

Description: Przechowuje informacje o wycieczkach

- **TourID** - Unikalny identyfikator wycieczki, typ danych: int
- **Name** - Nazwa wycieczki, typ danych: char
- **Description** - Opis wycieczki, typ danych: varchar
- **Price** - Cena wycieczki, typ danych: money
- **Duration** - Czas trwania wycieczki w minutach, typ danych: int
- **Tours_ak_1** - Warunek integralności: unikalność nazwy wycieczki
- **Tours_pk** - Warunek integralności: klucz główny

SQL Code:

```
CREATE TABLE Tours (
  TourID int NOT NULL,
  Name char NOT NULL,
  Description varchar NOT NULL,
  Price money NOT NULL,
  Duration int NOT NULL,
  CONSTRAINT Tours_ak_1 UNIQUE (Name),
  CONSTRAINT Tours_pk PRIMARY KEY (TourID)
);
```

Table: Tours_Places

Description: Przechowuje informacje o miejscach wycieczek

- **TourID** - Identyfikator wycieczki, typ danych: int (FK)
- **PlaceID** - Identyfikator miejsca, typ danych: int (FK)
- **Tours_Places_pk** - Warunek integralności: klucz główny złożony

SQL Code:

```
CREATE TABLE Tours_Places (  
    TourID int NOT NULL,  
    PlaceID int NOT NULL,  
    CONSTRAINT Tours_Places_pk PRIMARY KEY (TourID, PlaceID)  
);
```

Table: Payments

Description: Przechowuje informacje o miejscach wycieczek

- **PaymentID** - Identyfikator wpłaty za zamówienie, typ danych: int (FK)
- **OrderID** - Identyfikator zamówienia, typ danych: int (FK)
- **Amount** - Kwota wpłacona: money
- **Date** - Data wpłaty: date

SQL Code:

```
CREATE TABLE Payments (  
    PaymentID int NOT NULL,  
    OrderID int NOT NULL,  
    Amount money NOT NULL,  
    Date date NOT NULL,  
    CONSTRAINT Payments_pk PRIMARY KEY (PaymentID)  
);
```

Referencje

```
-- Reference: Employees_Places (table: Employees)  
ALTER TABLE Employees ADD CONSTRAINT Employees_Places  
    FOREIGN KEY (PlaceID)  
    REFERENCES Places (PlaceID);  
  
-- Reference: Employees_Positions (table: Employees)  
ALTER TABLE Employees ADD CONSTRAINT Employees_Positions  
    FOREIGN KEY (PositionID)  
    REFERENCES Positions (PositionID);  
  
-- Reference: Firms_Clients_Clients (table: Comapnies_Clients)  
ALTER TABLE Comapnies_Clients ADD CONSTRAINT Firms_Clients_Clients  
    FOREIGN KEY (ClientID)  
    REFERENCES Clients (ClientID);  
  
-- Reference: Firms_Clients_Firms (table: Comapnies_Clients)  
ALTER TABLE Comapnies_Clients ADD CONSTRAINT Firms_Clients_Firms  
    FOREIGN KEY (ComapnyID)
```

```

REFERENCES Comapnies (CompanyID);

-- Reference: Order_Atractions_Orders (table: Order_Atractions)
ALTER TABLE Order_Atractions ADD CONSTRAINT Order_Atractions_Orders
FOREIGN KEY (OrderID)
REFERENCES Orders (OrderID);

-- Reference: Order_Atractions_Participants (table: Order_Atractions)
ALTER TABLE Order_Atractions ADD CONSTRAINT
Order_Atractions_Participants
FOREIGN KEY (ParticipantsID)
REFERENCES Participants (ParticipantsID);

-- Reference: Order_Participants_Orders (table: Order_Participants)
ALTER TABLE Order_Participants ADD CONSTRAINT Order_Participants_Orders
FOREIGN KEY (OrderID)
REFERENCES Orders (OrderID);

-- Reference: Order_Participants_Participants (table:
Order_Participants)
ALTER TABLE Order_Participants ADD CONSTRAINT
Order_Participants_Participants
FOREIGN KEY (ParticipantsID)
REFERENCES Participants (ParticipantsID);

-- Reference: Order_Services_Orders (table: Order_Services)
ALTER TABLE Order_Services ADD CONSTRAINT Order_Services_Orders
FOREIGN KEY (OrderID)
REFERENCES Orders (OrderID);

-- Reference: Order_Services_Participants (table: Order_Services)
ALTER TABLE Order_Services ADD CONSTRAINT Order_Services_Participants
FOREIGN KEY (ParticipantsID)
REFERENCES Participants (ParticipantsID);

-- Reference: Orders_Clients (table: Orders)
ALTER TABLE Orders ADD CONSTRAINT Orders_Clients
FOREIGN KEY (ClientID)
REFERENCES Clients (ClientID);

-- Reference: Orders_Employees (table: Orders)
ALTER TABLE Orders ADD CONSTRAINT Orders_Employees
FOREIGN KEY (EmployeeID)
REFERENCES Employees (EmployeeID);

-- Reference: Orders_OrderStatus (table: Orders)

```

```
ALTER TABLE Orders ADD CONSTRAINT Orders_OrderStatus
    FOREIGN KEY (OrderStatusID)
    REFERENCES OrderStatus (OrderStatusID);

-- Reference: Orders_Services_Services (table: Order_Services)
ALTER TABLE Order_Services ADD CONSTRAINT Orders_Services_Services
    FOREIGN KEY (ServiceID)
    REFERENCES Services (ServiceID);

-- Reference: Orders_Tours (table: Orders)
ALTER TABLE Orders ADD CONSTRAINT Orders_Tours
    FOREIGN KEY (TourID)
    REFERENCES Tours (TourID);

-- Reference: Participants_Atractions_Atractions (table:
Order_Atractions)
ALTER TABLE Order_Atractions ADD CONSTRAINT
Participants_Atractions_Atractions
    FOREIGN KEY (AttractionID)
    REFERENCES Attractions (AttractionID);

-- Reference: ReportsTo (table: Employees)
ALTER TABLE Employees ADD CONSTRAINT ReportsTo
    FOREIGN KEY (ReportsTo)
    REFERENCES Employees (EmployeeID);

-- Reference: Services_Tours (table: Services)
ALTER TABLE Services ADD CONSTRAINT Services_Tours
    FOREIGN KEY (TourID)
    REFERENCES Tours (TourID);

-- Reference: Tours_Places_Places (table: Tours_Places)
ALTER TABLE Tours_Places ADD CONSTRAINT Tours_Places_Places
    FOREIGN KEY (PlaceID)
    REFERENCES Places (PlaceID);

-- Reference: Tours_Places_Tours (table: Tours_Places)
ALTER TABLE Tours_Places ADD CONSTRAINT Tours_Places_Tours
    FOREIGN KEY (TourID)
    REFERENCES Tours (TourID);

ALTER TABLE Payments ADD CONSTRAINT Payments_Orders
    FOREIGN KEY (OrderID)
    REFERENCES Orders (OrderID);
```

Wygenerowane dane testowe

Wszystkie dane testowe zostały wygenerowane na podstawie, wyżej wypisanych schematów. Do ich wygenerowania wykorzystaliśmy **Chat GPT 4o**

Atractions

```
INSERT INTO Atractions (AttractionID, Name, Price, Description,
MaxAtendents, TourID) VALUES
(1, 'Museum Tour', 50.00, 'A guided tour of the city museum.', 30, 1),
(2, 'Boat Ride', 30.00, 'A relaxing boat ride on the river.', 20, 2),
(3, 'Historical Walk', 40.00, 'A walk through the historical parts of
the city.', 25, 3),
(4, 'City Bike Tour', 20.00, 'A bike tour around the city.', 15, 4),
(5, 'Art Gallery Visit', 35.00, 'A visit to the modern art gallery.',
20, 1),
(6, 'Wine Tasting', 60.00, 'A wine tasting experience.', 10, 2),
(7, 'Night City Tour', 45.00, 'A guided tour of the city at night.', 25,
3),
(8, 'Mountain Hike', 70.00, 'A hike in the nearby mountains.', 15, 4),
(9, 'Beach Day', 20.00, 'A relaxing day at the beach.', 50, 2),
(10, 'Safari Adventure', 100.00, 'A safari adventure in the national
park.', 20, 3);
```

Cities

```
INSERT INTO Cities (CityID, City_Name, CountryID) VALUES
(1, 'Warsaw', 1),
(2, 'Krakow', 1),
(3, 'Berlin', 2),
(4, 'Munich', 2),
(5, 'Paris', 3),
(6, 'Lyon', 3),
(7, 'Madrid', 4),
(8, 'Barcelona', 4),
(9, 'Rome', 5),
(10, 'Milan', 5);
```

Clients

```
INSERT INTO Clients (ClientID, FirstName, LastName, Email, PhoneNumber,
BirthDate, PostalCode, Address, PlaceID) VALUES
(1, 'John', 'Doe', 'john.doe@example.com', '123456789', '1985-06-15',
'00-001', 'Main St 123', 1),
(2, 'Jane', 'Smith', 'jane.smith@example.com', '987654321', '1990-07-
```

```

20', '00-002', 'High St 456', 2),
(3, 'Alice', 'Johnson', 'alice.johnson@example.com', '555666777', '1982-
04-10', '00-003', 'Maple St 789', 3),
(4, 'Bob', 'Brown', 'bob.brown@example.com', '444333222', '1975-03-05',
'00-004', 'Oak St 101', 4),
(5, 'Charlie', 'Davis', 'charlie.davis@example.com', '333222111', '1988-
12-25', '00-005', 'Pine St 202', 5),
(6, 'Diana', 'Clark', 'diana.clark@example.com', '111333444', '1995-03-
15', '00-006', 'Cedar St 303', 6),
(7, 'Evan', 'Miller', 'evan.miller@example.com', '666777888', '1980-11-
05', '00-007', 'Birch St 404', 7),
(8, 'Fiona', 'Wilson', 'fiona.wilson@example.com', '999888777', '1972-
09-18', '00-008', 'Elm St 505', 8),
(9, 'George', 'White', 'george.white@example.com', '555444333', '1965-
04-22', '00-009', 'Ash St 606', 9),
(10, 'Hannah', 'Taylor', 'hannah.taylor@example.com', '222111000',
'1999-08-30', '00-010', 'Fir St 707', 10);

```

Comapnies_Clients

```

INSERT INTO Comapnies_Clients (ComapnyID, ClientID) VALUES
(1, 1),
(1, 2),
(2, 3),
(2, 4),
(3, 5),
(3, 6),
(4, 7),
(4, 8),
(5, 9),
(5, 10);

```

Companies

```

INSERT INTO Companies (CompanyID, Name, Address, NIP, REGON,
PhoneNumber, Email, PlaceID) VALUES
(1, 'TravelCo', 'Main St 123', '123-456-32-18', '12345678901234',
'111222333', 'info@travelco.com', 1),
(2, 'AdventureWorks', 'High St 456', '456-789-54-32', '43210987654321',
'444555666', 'contact@adventureworks.com', 2),
(3, 'ExploreMore', 'Maple St 789', '789-012-76-54', '56789012345678',
'777888999', 'info@exploremore.com', 3),
(4, 'Wanderlust', 'Oak St 101', '101-234-98-76', '87654321098765',

```

```
'222333444', 'hello@wanderlust.com', 4),
(5, 'JourneyMakers', 'Pine St 202', '202-345-21-98', '34567890123456',
'333444555', 'info@journeymakers.com', 5),
(6, 'VoyageVentures', 'Cedar St 303', '303-456-43-10', '23456789012345',
'666777888', 'contact@voyageventures.com', 6),
(7, 'QuestQuest', 'Birch St 404', '404-567-65-32', '12345678901234',
'999000111', 'info@questquest.com', 7),
(8, 'GlobeTrotters', 'Elm St 505', '505-678-87-54', '56789012345678',
'444555666', 'contact@globetrotters.com', 8),
(9, 'TravelMasters', 'Ash St 606', '606-789-09-76', '87654321098765',
'777888999', 'info@travelmasters.com', 9),
(10, 'WorldWonders', 'Fir St 707', '707-890-21-98', '34567890123456',
'888999000', 'hello@worldwonders.com', 10);
```

Countries

```
INSERT INTO Countries (CountryID, Country_Name) VALUES
(1, 'Poland'),
(2, 'Germany'),
(3, 'France'),
(4, 'Spain'),
(5, 'Italy'),
(6, 'United Kingdom'),
(7, 'Netherlands'),
(8, 'Belgium'),
(9, 'Austria'),
(10, 'Switzerland');
```

Employee_Attraction

```
INSERT INTO Employee_Attraction (EmployeeID, AtractionID) VALUES
(1, 1),
(2, 2),
(3, 3),
(4, 4),
(5, 5),
(6, 6),
(7, 7),
(8, 8),
(9, 9),
(10, 10),
(1, 2),
(2, 3),
(3, 4),
```

```
(4, 5),  
(5, 6);
```

Employee_Service

```
INSERT INTO Employee_Service (EmployeeID, ServiceID) VALUES  
(1, 1),  
(2, 2),  
(3, 3),  
(4, 4),  
(5, 1),  
(6, 2),  
(7, 3),  
(8, 4),  
(9, 1),  
(10, 2),  
(1, 3),  
(2, 4),  
(3, 1),  
(4, 2),  
(5, 3);
```

Employee_Tour

```
INSERT INTO Employee_Tour (EmployeeID, TourID) VALUES  
(1, 1),  
(2, 2),  
(3, 3),  
(4, 4),  
(5, 1),  
(6, 2),  
(7, 3),  
(8, 4),  
(9, 1),  
(10, 2),  
(1, 3),  
(2, 4),  
(3, 1),  
(4, 2),  
(5, 3);
```

Employees

```

INSERT INTO Employees (EmployeeID, FirstName, LastName, BirthDate,
HireDate, Address, PostalCode, PhoneNumber, Email, Extension, Photo,
ReportsTo, PositionID, PlaceID) VALUES
(1, 'Tom', 'Harris', '1980-01-01', '2020-01-15', 'Main St 123', '00-001', '123456789', 'tom.harris@example.com', '1234', 0x00, NULL, 1, 1),
(2, 'Sara', 'Connor', '1985-02-02', '2019-03-10', 'High St 456', '00-002', '987654321', 'sara.connor@example.com', '5678', 0x00, 1, 2, 2),
(3, 'Mike', 'Tyson', '1990-03-03', '2021-05-20', 'Maple St 789', '00-003', '555666777', 'mike.tyson@example.com', '9012', 0x00, 2, 3, 3),
(4, 'Lisa', 'Simpson', '1995-04-04', '2018-11-30', 'Oak St 101', '00-004', '444333222', 'lisa.simpson@example.com', '3456', 0x00, 3, 4, 4),
(5, 'Jake', 'Peralta', '1980-05-05', '2017-07-25', 'Pine St 202', '00-005', '333222111', 'jake.peralta@example.com', '7890', 0x00, 4, 5, 5),
(6, 'Amy', 'Santiago', '1985-06-06', '2016-05-14', 'Cedar St 303', '00-006', '111333444', 'amy.santiago@example.com', '1234', 0x00, 5, 6, 6),
(7, 'Terry', 'Jeffords', '1990-07-07', '2015-03-09', 'Birch St 404', '00-007', '666777888', 'terry.jeffords@example.com', '5678', 0x00, 6, 7, 7),
(8, 'Rosa', 'Diaz', '1995-08-08', '2014-09-18', 'Elm St 505', '00-008', '999888777', 'rosa.diaz@example.com', '9012', 0x00, 7, 8, 8),
(9, 'Gina', 'Linetti', '1980-09-09', '2013-11-22', 'Ash St 606', '00-009', '555444333', 'gina.linetti@example.com', '3456', 0x00, 8, 9, 9),
(10, 'Charles', 'Boyle', '1985-10-10', '2012-08-16', 'Fir St 707', '00-010', '222111000', 'charles.boyle@example.com', '7890', 0x00, 9, 10, 10),
(11, 'Holt', 'Raymond', '1965-11-11', '2011-07-10', 'Ninth St 909', '00-011', '111222333', 'holt.raymond@example.com', '4321', 0x00, NULL, 1, 11),
(12, 'Hitchcock', 'Michael', '1970-12-12', '1990-09-20', 'Brooklyn St 808', '00-012', '222333444', 'hitchcock.michael@example.com', '8765', 0x00, 11, 2, 12);

```

OrderStatus

```

INSERT INTO OrderStatus (OrderStatusID, OrderStatus) VALUES
(1, 'Pending'),
(2, 'Confirmed'),
(3, 'Shipped'),
(4, 'Delivered'),
(5, 'Cancelled'),
(6, 'Returned');

```

Order_Atractions


```
INSERT INTO Order_Atractions (AttractionID, OrderID, ParticipantsID)
VALUES
(1, 1, 1),
(2, 2, 2),
(3, 3, 3),
(4, 4, 4),
(5, 5, 5),
(6, 6, 6),
(7, 7, 7),
(8, 8, 8),
(9, 9, 9),
(10, 10, 10),
(1, 2, 3),
(2, 3, 4),
(3, 4, 5),
(4, 5, 6),
(5, 6, 7);
```

Order_Participants

```
INSERT INTO Order_Participants (OrderID, ParticipantsID) VALUES
(1, 1),
(2, 2),
(3, 3),
(4, 4),
(5, 5),
(6, 6),
(7, 7),
(8, 8),
(9, 9),
(10, 10),
(1, 2),
(2, 3),
(3, 4),
(4, 5),
(5, 6);
```

Order_Services

```
INSERT INTO Order_Services (ServiceID, OrderID, ParticipantsID) VALUES
(1, 1, 1),
(2, 2, 2),
(3, 3, 3),
(4, 4, 4),
```

```
(5, 1, 5),
(6, 2, 6),
(7, 3, 7),
(8, 4, 8),
(9, 1, 9),
(10, 2, 10),
(1, 3, 1),
(2, 4, 2),
(3, 5, 3),
(4, 6, 4),
(5, 7, 5);
```

Orders

```
INSERT INTO Orders (OrderID, ClientID, OrderDate, OrderStatusID, TourID,
Discount, PaymentDue, EmployeeID) VALUES
(1, 1, '2023-01-01', 1, 1, 10.00, '2023-02-01', 1),
(2, 2, '2023-02-15', 2, 2, 15.00, '2023-03-15', 2),
(3, 3, '2023-03-20', 3, 3, 20.00, '2023-04-20', 3),
(4, 4, '2023-04-25', 4, 4, 25.00, '2023-05-25', 4),
(5, 5, '2023-05-30', 5, 1, 30.00, '2023-06-30', 5),
(6, 6, '2023-06-10', 6, 2, 35.00, '2023-07-10', 6),
(7, 7, '2023-07-15', 1, 3, 40.00, '2023-08-15', 7),
(8, 8, '2023-08-20', 2, 4, 45.00, '2023-09-20', 8),
(9, 9, '2023-09-25', 3, 1, 50.00, '2023-10-25', 9),
(10, 10, '2023-10-30', 4, 2, 55.00, '2023-11-30', 10),
(11, 1, '2023-11-01', 5, 3, 60.00, '2023-12-01', 11),
(12, 2, '2023-12-15', 6, 4, 65.00, '2024-01-15', 12);
```

Participants

```
INSERT INTO Participants (ParticipantsID, FirstName, LastName,
BirthDate) VALUES
(1, 'Tim', 'Cook', '1990-05-01'),
(2, 'Elon', 'Musk', '1980-06-28'),
(3, 'Jeff', 'Bezos', '1964-01-12'),
(4, 'Bill', 'Gates', '1955-10-28'),
(5, 'Mark', 'Zuckerberg', '1984-05-14'),
(6, 'Larry', 'Page', '1973-03-26'),
(7, 'Sergey', 'Brin', '1973-08-21'),
(8, 'Steve', 'Jobs', '1955-02-24'),
(9, 'Sundar', 'Pichai', '1972-07-12'),
(10, 'Satya', 'Nadella', '1967-08-19'),
(11, 'Tim', 'Berners-Lee', '1955-06-08'),
```

```
(12, 'Vint', 'Cerf', '1943-06-23');
```

Places

```
INSERT INTO Places (PlaceID, CityID) VALUES
(1, 1),
(2, 2),
(3, 3),
(4, 4),
(5, 5),
(6, 6),
(7, 7),
(8, 8),
(9, 9),
(10, 10),
(11, 1),
(12, 2),
(13, 3),
(14, 4),
(15, 5);
```

Positions

```
INSERT INTO Positions (PositionID, JobName) VALUES
(1, 'Tour Guide'),
(2, 'Manager'),
(3, 'Driver'),
(4, 'Receptionist'),
(5, 'Coordinator'),
(6, 'Sales Representative'),
(7, 'Marketing Specialist'),
(8, 'IT Support'),
(9, 'Accountant'),
(10, 'HR Manager');
```

Services

```
INSERT INTO Services (ServiceID, Name, Price, Description, MaxAttendents,
TourID) VALUES
(1, 'Guided Tour', 100.00, 'A full day guided tour.', 30, 1),
(2, 'Transport', 50.00, 'Transportation services.', 20, 2),
(3, 'Accommodation', 200.00, 'Hotel accommodation.', 15, 3),
(4, 'Meal', 30.00, 'Meal services.', 50, 4),
(5, 'Photography', 80.00, 'Professional photography services.', 10, 1),
```

```
(6, 'Translation', 40.00, 'Translation and interpretation services.',
20, 2),
(7, 'VIP Package', 500.00, 'Exclusive VIP package.', 5, 3),
(8, 'Insurance', 20.00, 'Travel insurance.', 100, 4),
(9, 'Event Ticket', 60.00, 'Ticket to a local event.', 30, 1),
(10, 'Souvenir Package', 25.00, 'Package of local souvenirs.', 50, 2),
(11, 'Excursion', 150.00, 'Half-day excursion.', 25, 3),
(12, 'Workshop', 70.00, 'Interactive workshop.', 15, 4);
```

Tours

```
INSERT INTO Tours (TourID, TourName, Description, MaxAttendents, Price,
TourDateBegines, TourDateEnds, TourLanguage) VALUES
(1, 'City Tour', 'A tour around the city.', 30, 150.00, '2023-01-01',
'2023-01-02', 'English'),
(2, 'Historical Tour', 'A tour of historical sites.', 20, 200.00, '2023-
02-01', '2023-02-02', 'Spanish'),
(3, 'Cultural Tour', 'A tour of cultural landmarks.', 25, 180.00, '2023-
03-01', '2023-03-02', 'French'),
(4, 'Adventure Tour', 'An adventure tour.', 15, 220.00, '2023-04-01',
'2023-04-02', 'German'),
(5, 'Food Tour', 'A tour of local cuisine.', 20, 130.00, '2023-05-01',
'2023-05-02', 'Italian'),
(6, 'Nightlife Tour', 'A tour of the city\'s nightlife.', 30, 170.00,
'2023-06-01', '2023-06-02', 'Portuguese'),
(7, 'Nature Tour', 'A tour of natural attractions.', 10, 160.00, '2023-
07-01', '2023-07-02', 'Dutch'),
(8, 'Art Tour', 'A tour of local art galleries.', 25, 140.00, '2023-08-
01', '2023-08-02', 'Japanese'),
(9, 'Shopping Tour', 'A tour of the best shopping spots.', 40, 110.00,
'2023-09-01', '2023-09-02', 'Chinese'),
(10, 'Luxury Tour', 'An exclusive luxury tour.', 5, 300.00, '2023-10-
01', '2023-10-02', 'Russian'),
(11, 'Music Tour', 'A tour of local music venues.', 20, 175.00, '2023-
11-01', '2023-11-02', 'Korean'),
(12, 'Theater Tour', 'A tour of historic theaters.', 15, 190.00, '2023-
12-01', '2023-12-02', 'Italian');
```

Tours_Places

```
INSERT INTO Tours_Places (PlaceID, TourID) VALUES
(1, 1),
(2, 2),
(3, 3),
```

```
(4, 4),  
(5, 5),  
(6, 6),  
(7, 7),  
(8, 8),  
(9, 9),  
(10, 10),  
(11, 1),  
(12, 2),  
(13, 3),  
(14, 4),  
(15, 5);
```

Payments

Zostały wygenerowane na podstawie Orders i widoku OrderFinalPrice

```
-- Wstawianie przykładowych danych do tabeli Payments  
INSERT INTO Payments (PaymentID, OrderID, Amount, Date) VALUES  
(1, 2, 515.00, '2023-03-01'), -- Fully paid in one payment  
(2, 3, 445.00, '2023-04-01'),  
(3, 3, 445.00, '2023-04-15'),  
(4, 4, 252.50, '2023-05-01'),  
(5, 4, 252.50, '2023-05-15'),  
(6, 5, 325.00, '2023-06-10'), -- Fully paid in one payment  
(7, 6, 112.50, '2023-06-15'),  
(8, 6, 112.50, '2023-07-01'),  
(9, 8, 122.50, '2023-09-01'),  
(10, 8, 122.50, '2023-09-15'),  
(11, 9, 60.00, '2023-10-01'),  
(12, 9, 60.00, '2023-10-20'),  
(13, 10, 122.50, '2023-11-05'),  
(14, 10, 122.50, '2023-11-20'),  
(15, 15, 150.00, '2023-02-10'), -- Fully paid in one payment  
(16, 16, 75.00, '2023-02-10'),  
(17, 16, 75.00, '2023-02-20');
```

Widoki

1. Cena za dane zamówienie

Widok wyświetla łączną kwotę, jaka wyszła za daną wycieczkę.(Karol)

```

CREATE VIEW OrderFinalPrice AS
SELECT
    o.OrderID,
    SUM(t.Price + COALESCE(a.TotalAttractionCost, 0) +
COALESCE(s.TotalServiceCost, 0)) * (1 - COALESCE(o.Discount, 0)) AS
FinalPrice
FROM
    Orders o
JOIN
    Order_Participants op ON o.OrderID = op.OrderID
JOIN
    Tours t ON o.TourID = t.TourID
LEFT JOIN
    (
        SELECT
            oa.OrderID,
            oa.ParticipantID,
            SUM(a.Price) AS TotalAttractionCost
        FROM
            Order_Attractions oa
        JOIN
            Attractions a ON oa.AttractionID = a.AttractionID
        GROUP BY
            oa.OrderID, oa.ParticipantID
    ) a ON op.OrderID = a.OrderID AND op.ParticipantID = a.ParticipantID
LEFT JOIN
    (
        SELECT
            os.OrderID,
            os.ParticipantID,
            SUM(s.Price) AS TotalServiceCost
        FROM
            Order_Services os
        JOIN
            Services s ON os.ServiceID = s.ServiceID
        GROUP BY
            os.OrderID, os.ParticipantID
    ) s ON op.OrderID = s.OrderID AND op.ParticipantID = s.ParticipantID
GROUP BY
    o.OrderID, o.Discount

```

Przykład użycia:

1

```
select * from [OrderFinalPrice]
```

Results

Messages

	OrderID ▾	FinalPrice ▾
1	1	440,00
2	2	515,00
3	3	610,00
4	4	505,00
5	5	325,00
6	6	225,00
7	7	185,00
8	8	245,00
9	9	120,00
10	10	245,00
11	15	150,00

2. Uczestnicy wycieczki

Widok wyświetla wszystkich uczestników, którzy zostali zapisani do danej wycieczki.(Karol)

```
CREATE VIEW TourParticipants AS
SELECT
    t.TourID,
    t.TourName,
    p.ParticipantID,
    p.FirstName,
    p.LastName,
    p.BirthDate
FROM
    Tours t
JOIN
    Orders o ON t.TourID = o.TourID
JOIN
    Order_Participants op ON o.OrderID = op.OrderID
```

JOIN

Participants p ON op.ParticipantID = p.ParticipantID

Przykład użycia:

1 select * from TourParticipants

Results Messages

	TourID	TourName	ParticipantsID	FirstName	LastName
1	1	City Tour	1	Tim	Cook
2	1	City Tour	2	Elon	Musk
3	2	Historical Tour	2	Elon	Musk
4	2	Historical Tour	3	Jeff	Bezos
5	3	Cultural Tour	3	Jeff	Bezos
6	3	Cultural Tour	4	Bill	Gates
7	4	Adventure Tour	4	Bill	Gates
8	4	Adventure Tour	5	Mark	Zuckerberg
9	1	City Tour	5	Mark	Zuckerberg
10	1	City Tour	6	Larry	Page
11	2	Historical Tour	6	Larry	Page
12	3	Cultural Tour	7	Sergey	Brin
13	4	Adventure Tour	8	Steve	Jobs
14	1	City Tour	9	Sundar	Pichai
15	2	Historical Tour	10	Satya	Nadella
16	1	City Tour	1	Tim	Cook

3. Najczęściej wybierane wycieczki

Widok wyświetla 3 najczęściej wybierane wycieczki.(Karol)

```
CREATE VIEW Top3Tours AS
SELECT
    TOP 3
    t.TourID,
    t.TourName,
    COUNT(o.OrderID) AS OrderCount
FROM
    Tours t
JOIN
    Orders o ON t.TourID = o.TourID
GROUP BY
    t.TourID,
    t.TourName
```

Przykład użycia:

1
select * from Top3Tours

Results
Messages

	TourID	TourName	OrderCount
1	1	City Tour	6
2	2	Historical Tour	3
3	3	Cultural Tour	3

4. Najczęściej wybierane atrakcje

Widok wyświetla najczęściej wybierane atrakcje w kontekście danej wycieczki.(Mieszko)

```
CREATE VIEW TopAttractionPerTour AS
WITH AttractionCounts AS (
    SELECT
        t.TourID,
        t.TourName,
        a.AttractionID,
        a.Name AS AttractionName,
        COUNT(oa.ParticipantID) AS ParticipantCount,
        ROW_NUMBER() OVER (PARTITION BY t.TourID ORDER BY
COUNT(oa.ParticipantID) DESC) AS rn
    FROM
        Tours t
    JOIN
        Attractions a ON t.TourID = a.TourID
    JOIN
        Order_Attractions oa ON a.AttractionID = oa.AttractionID
    GROUP BY
        t.TourID,
        t.TourName,
        a.AttractionID,
        a.Name
)
SELECT
    TourID,
    TourName,
    AttractionID,
    AttractionName,
    ParticipantCount
FROM
    AttractionCounts
WHERE
    rn = 1
```

Przykład użycia:

1

select * from TopAttractionPerTour

Results

Messages

	TourID	TourName	AttractionID	AttractionName	ParticipantCount
1	1	City Tour	1	Museum Tour	2
2	2	Historical Tour	2	Boat Ride	2
3	3	Cultural Tour	3	Historical Walk	2
4	4	Adventure Tour	4	City Bike Tour	2

5. Najczęściej wybierane serwisy

Widok wyświetla najczęściej wybierane serwisy w kontekście danej wycieczki.(Mieszko)

```
CREATE VIEW TopServicePerTour AS
WITH ServiceCounts AS (
    SELECT
        t.TourID,
        t.TourName,
        s.ServiceID,
        s.Name AS ServiceName,
        COUNT(os.ParticipantID) AS ParticipantCount,
        ROW_NUMBER() OVER (PARTITION BY t.TourID ORDER BY
COUNT(os.ParticipantID) DESC) AS rn
    FROM
        Tours t
    JOIN
        Services s ON t.TourID = s.TourID
    JOIN
        Order_Services os ON s.ServiceID = os.ServiceID
    GROUP BY
        t.TourID,
        t.TourName,
        s.ServiceID,
        s.Name
)
SELECT
    TourID,
    TourName,
    ServiceID,
    ServiceName,
    ParticipantCount
FROM
    ServiceCounts
WHERE
    rn = 1
```

Przykład użycia:

```
1 select * from TopServicePerTour
```

Results					
	TourID	TourName	ServiceID	ServiceName	ParticipantCount
1	1	City Tour	1	Guided Tour	2
2	2	Historical Tour	2	Transport	2
3	3	Cultural Tour	3	Accommodation	2
4	4	Adventure Tour	4	Meal	2

6. Zamówienia oczekujące na zapłatę

Widok pokazujący wszystkie zamówienia, które mają status "Pending" i wymagają płatności, wraz z terminami płatności i kwotą do zapłaty. (Mateusz)

```
CREATE VIEW Pending_Payments AS
SELECT
    O.OrderID,
    O.ClientID,
    C.FirstName,
    C.LastName,
    O.OrderDate,
    T.TourName,
    O.Discount,
    O.PaymentDue,
    O.EmployeeID,
    (T.Price + ISNULL(SUM(A.Price), 0) - O.Discount) AS AmountDue
FROM
    Orders O
    INNER JOIN Clients C ON O.ClientID = C.ClientID
    INNER JOIN OrderStatus OS ON O.OrderStatusID = OS.OrderStatusID
    INNER JOIN Tours T ON O.TourID = T.TourID
    LEFT JOIN Order_Atractions OA ON O.OrderID = OA.OrderID
    LEFT JOIN Atractions A ON OA.AtractionID = A.AtractionID
WHERE
    OS.OrderStatus = 'Pending'
GROUP BY
    O.OrderID,
    O.ClientID,
    C.FirstName,
    C.LastName,
    O.OrderDate,
    T.TourName,
    O.Discount,
    O.PaymentDue,
    O.EmployeeID,
    T.Price;
```

GO

Przykład użycia:

1

select * from [Pending_Payments]

Results

Messages

	OrderID	ClientID	FirstName	LastName	OrderDate	TourName
1	1	1	John	Doe	2023-01-01	City Tour
2	7	7	Evan	Miller	2023-07-15	Cultural Tour
3	13	1	John	Doe	2024-05-27	City Tour
4	14	1	John	Doe	2024-06-09	City Tour
5	16	1	John	Doe	2023-01-01	City Tour

7. Najwyższe zapotrzebowanie na wycieczki

Widok pokazujący okresy o najwyższym zapotrzebowaniu na wycieczki, uwzględniając liczbę rezerwacji (bez 'Cancelled' i 'Returned') w określonych miesiącach i latach. (Mateusz)

```
CREATE VIEW High_Demand_Periods AS
SELECT
    YEAR(T.TourDateBegins) AS Year,
    MONTH(T.TourDateBegins) AS Month,
    COUNT(OP.ParticipantsID) AS TotalParticipants
FROM
    Orders O
    INNER JOIN Tours T ON O.TourID = T.TourID
    INNER JOIN Order_Participants OP ON O.OrderID = OP.OrderID
    INNER JOIN OrderStatus OS ON O.OrderStatusID = OS.OrderStatusID
WHERE
    OS.OrderStatus NOT IN ('Cancelled', 'Returned')
GROUP BY
    YEAR(T.TourDateBegins), MONTH(T.TourDateBegins)
ORDER BY
    TotalParticipants DESC;
```

GO

Przykład użycia:

```
1 select * from [High_Demand_Periods]
```

Results Messages

	Year	Month	TotalParticipants
1	2023	1	5
2	2023	2	3
3	2023	3	3
4	2023	4	3

8. Nieopłacone rezerwacje na tydzień przed

Widok pokazuje nieopłacone zamówienia, którym pozostaje mniej niż tydzień do rozpoczęcia wycieczki. (Patryk)

```
CREATE VIEW Unpaid_Orders_One_Week_Left AS
SELECT
    O.OrderID,
    O.ClientID,
    O.OrderDate,
    O.TourID,
    T.TourName,
    T.TourDateBegines,
    O.Discount,
    O.PaymentDue,
    O.EmployeeID,
    DATEDIFF(day, GETDATE(), T.TourDateBegines) AS DaysLeft
FROM
    Orders O
    INNER JOIN Tours T ON O.TourID = T.TourID
    INNER JOIN OrderStatus OS ON O.OrderStatusID = OS.OrderStatusID
WHERE
    OS.OrderStatus = 'Pending'
    AND DATEDIFF(day, GETDATE(), T.TourDateBegines) <= 7;
GO
```

Przykład użycia:

1

select * from Unpaid_Orders_One_Week_Left

Results		Messages							
	OrderID	ClientID	OrderDate	TourID	TourName	TourDateBegins	Discount	PaymentDue	EmployeeID
1	1	1	2023-01-01	1	City Tour	2023-01-01	10,00	2023-02-01	1
2	7	7	2023-07-15	3	Cultural Tour	2023-03-01	40,00	2023-08-15	7
3	13	1	2024-05-27	1	City Tour	2023-01-01	10,00	2024-06-03	1
4	14	1	2024-06-09	1	City Tour	2023-01-01	10,00	2024-06-10	1
5	16	1	2023-01-01	1	City Tour	2023-01-01	0,00	2022-12-25	1

9. Historia zamówień klientów

Widok pokazujący historię zamówień poszczególnych klientów, w tym daty zamówień, statusy zamówień oraz nazwy wycieczek. (Patrik)

```
CREATE VIEW Client_Order_History AS
SELECT
    C.ClientID,
    C.FirstName,
    C.LastName,
    O.OrderID,
    O.OrderDate,
    OS.OrderStatus,
    T.TourName,
    O.Discount,
    O.PaymentDue,
    O.EmployeeID
FROM
    Clients C
    INNER JOIN Orders O ON C.ClientID = O.ClientID
    INNER JOIN OrderStatus OS ON O.OrderStatusID = OS.OrderStatusID
    INNER JOIN Tours T ON O.TourID = T.TourID;
GO
```

Przykład użycia:

1

select * from Client_Order_History

Results		Messages				
	ClientID	FirstName	LastName	OrderID	OrderDate	OrderStatus
1	1	John	Doe	1	2023-01-01	Pending
2	2	Jane	Smith	2	2023-02-15	Confirmed
3	3	Alice	Johnson	3	2023-03-20	Shipped
4	4	Bob	Brown	4	2023-04-25	Delivered
5	5	Charlie	Davis	5	2023-05-30	Cancelled
6	6	Diana	Clark	6	2023-06-10	Returned
7	7	Evan	Miller	7	2023-07-15	Pending
8	8	Fiona	Wilson	8	2023-08-20	Confirmed
9	9	George	White	9	2023-09-25	Shipped
10	10	Hannah	Taylor	10	2023-10-30	Delivered
11	1	John	Doe	11	2023-11-01	Cancelled
12	2	Jane	Smith	12	2023-12-15	Returned
13	1	John	Doe	13	2024-05-27	Pending

10. Dostępne wolne miejsca na wycieczce

Widok pokazuje, ile wolnych miejsc pozostało na każdą wycieczkę, uwzględniając maksymalną liczbę uczestników oraz liczbę już zarezerwowanych miejsc. (Patryk)

```
CREATE VIEW Available_Tour_Spots AS
SELECT
    T.TourID,
    T.TourName,
    T.MaxAtendents AS MaxParticipants,
    ISNULL(SUM(OP.Participants), 0) AS ReservedSpots,
    T.MaxAtendents - ISNULL(SUM(OP.Participants), 0) AS AvailableSpots
FROM
    Tours T
    LEFT JOIN Orders O ON T.TourID = O.TourID
    LEFT JOIN Order_Participants OP ON O.OrderID = OP.OrderID
GROUP BY
    T.TourID,
    T.TourName,
    T.MaxAtendents;
GO
```

Przykład użycia:

1 `select * from [Available_Tour_Spots]`

Results

Messages

	TourID	TourName	Country_Name	City_Name
1	1	City Tour	Poland	Warsaw
2	2	Historical Tour	Poland	Krakow
3	3	Cultural Tour	Germany	Berlin
4	4	Adventure Tour	Germany	Munich
5	5	Food Tour	France	Paris
6	6	Nightlife Tour	France	Lyon
7	7	Nature Tour	Spain	Madrid
8	8	Art Tour	Spain	Barcelona
9	9	Shopping Tour	Italy	Rome
10	10	Luxury Tour	Italy	Milan

Procedury

1. Aktualizacja daty zapłaty za zamówienie

Procedura UpdatePaymentDueDate przyjmuje jako argument OrderID i aktualizuje kolumnę PaymentDue na bieżącą datę. (Karol)

```
CREATE PROCEDURE UpdatePaymentDueDate
    @OrderID INT
AS
```

```

BEGIN
    SET NOCOUNT ON;
    UPDATE Orders
    SET PaymentDue = GETDATE()
    WHERE OrderID = @OrderID;
END;
GO

```

Wynik:

The screenshot shows the SQL Server Enterprise Manager interface. The left pane displays the 'Database Explorer' for the 'AGH' database. The central pane shows the execution of a T-SQL script in the 'console_13' window. The script consists of three lines: 'EXEC UpdatePaymentDueDate @OrderID = 14;', 'SELECT * FROM Orders;', and a blank line. The right pane shows the 'Files' view. The bottom pane displays the 'Output' window, which shows the execution plan and the results of the 'SELECT * FROM Orders;' query. The results are displayed in a table with 16 rows and 7 columns: OrderID, ClientID, OrderDate, OrderStatusID, TourID, Discount, and PaymentDue. The table shows data for various orders, with the row for OrderID 14 highlighted in blue.

OrderID	ClientID	OrderDate	OrderStatusID	TourID	Discount	PaymentDue
12	2	2023-12-15	6	4	65.0000	2024-01-15
13	1	2024-05-27	1	1	10.0000	2024-06-03
14	1	2024-06-09	1	1	10.0000	2024-06-11
15	1	2023-01-01	2	1	0.0000	2024-06-11
16	1	2023-01-01	1	1	0.0000	2022-12-25

2. Wstawianie klienta jako nowego uczestnika wycieczki

Procedura AddParticipantToOrder dodaje rekord do tabeli Order_Participants. Przyjmuje dwa argumenty OrderID i ClientID. (Karol)

```

CREATE PROCEDURE AddParticipantToOrder
    @OrderID INT,
    @ClientID INT
AS
BEGIN
    SET NOCOUNT ON;
    INSERT INTO Order_Participants (OrderID, ParticipantsID)
    VALUES (@OrderID, @ClientID);

```



```
END;  
GO
```

Wynik:

The screenshot shows the SQL Server Enterprise Manager interface. The left pane displays the 'Database Explorer' for the 'b.li.agh.edu.pl' database, with the 'TourParticipants' table selected. The center pane shows the execution of a query: `EXEC AddParticipantToOrder @OrderID = 3, @ClientID = 1;` followed by `SELECT * FROM Order_Participants;`. The right pane shows the 'Files' view. The bottom pane displays the 'Output' window for the query 'u_wirkijow.dbo.Order_Participants', showing 18 rows of data. The data is presented in a table with columns 'OrderID' and 'ParticipantsID'.

OrderID	ParticipantsID
1	2
2	2
2	3
3	1
3	3
3	4
4	4
4	5

3. Sprawdzanie czy całość została już opłacona

Procedura **CheckAndUpdateOrderStatus** sprawdza, czy całkowita kwota wpłacona dla zamówienia jest równa wartości **FinalPrice**. Jeśli tak, aktualizuje status zamówienia na 2. (Patryk)

```
CREATE PROCEDURE CheckAndUpdateOrderStatus  
    @OrderID INT  
AS  
BEGIN  
    DECLARE @TotalPaid INT;  
    DECLARE @FinalPrice INT;  
  
    -- Oblicz całkowitą kwotę wpłaconą dla danego zamówienia  
    SELECT @TotalPaid = SUM(Amount)  
    FROM Payments  
    WHERE OrderID = @OrderID;  
  
    SELECT @FinalPrice = FinalPrice
```

```

FROM OrderFinalPrice
WHERE OrderID = @OrderID;

IF @TotalPaid >= @FinalPrice
BEGIN
    UPDATE Orders
    SET OrderStatusID = 2
    WHERE OrderID = @OrderID;
END
END;

```

4. Usuwanie nieopłaconych zamówień na tydzień

Ta procedura przechodzi przez wszystkie zamówienia w widoku **Unpaid_Orders_One_Week_Left** i aktualizuje ich status na "Cancelled", jeśli spełniają kryteria widoku. (Patryk)

```

CREATE PROCEDURE CheckAndCancelUnpaidOrders
AS
BEGIN
    DECLARE @OrderID INT;
    DECLARE OrderCursor CURSOR FOR
    SELECT OrderID FROM Unpaid_Orders_One_Week_Left;

    OPEN OrderCursor;

    FETCH NEXT FROM OrderCursor INTO @OrderID;

    WHILE @@FETCH_STATUS = 0
    BEGIN
        UPDATE Orders
        SET OrderStatusID = 5
        WHERE OrderID = @OrderID;

        FETCH NEXT FROM OrderCursor INTO @OrderID;
    END;

    CLOSE OrderCursor;
    DEALLOCATE OrderCursor;
END;

```

Triggery

1. SELECT * FROM Pending_Payments
2. Trigger na aktualizację stanu zamówienia:

Trigger, który automatycznie ustawia datę płatności na aktualną datę, gdy zamówienie jest oznaczone jako zakończone (np. "Completed"). (Karol)

```
CREATE TRIGGER SetPaymentDueOnComplete
ON Orders
AFTER UPDATE
AS
BEGIN
    DECLARE @OrderID INT
    SELECT @OrderID = i.OrderID
    FROM inserted i
    JOIN OrderStatus os ON i.OrderStatusID = os.OrderStatusID
    WHERE os.OrderStatus = 'Confirmed';
    EXEC UpdatePaymentDueDate @OrderID
END;
```

Wynik:

The screenshot displays the SQL Server Enterprise Manager interface. The left pane shows the 'Database Explorer' with the 'Orders' table selected under the 'OrderStatus' folder. The right pane shows the 'console_15' window with the following SQL script:

```
1 UPDATE Orders SET OrderStatusID=2 WHERE OrderID=16;
2 SELECT * FROM Orders WHERE OrderID=16;
```

The 'Output' pane at the bottom shows the results of the query, displaying a single row with the following data:

OrderID	ClientID	OrderDate	OrderStatusID	TourID	Discount	PaymentDue	Employee
16	1	2023-01-01	2	1	0.0000	2024-06-11	

3. Trigger na wstawienie nowego uczestnika:

Trigger, który automatycznie dodaje nowego uczestnika do odpowiedniej wycieczki w tabeli Order_Participants po utworzeniu zamówienia.(Karol)

```
CREATE TRIGGER AddParticipantToOrderTrigger
ON Orders
AFTER INSERT
AS
BEGIN
    SET NOCOUNT ON;
    DECLARE @OrderID INT, @ClientID INT;
    DECLARE InsertedCursor CURSOR FOR
    SELECT i.OrderID, i.ClientID
    FROM inserted i;
    OPEN InsertedCursor;
    FETCH NEXT FROM InsertedCursor INTO @OrderID, @ClientID;
    WHILE @@FETCH_STATUS = 0
    BEGIN
        EXEC AddParticipantToOrder @OrderID, @ClientID;
        FETCH NEXT FROM InsertedCursor INTO @OrderID, @ClientID;
    END;
    CLOSE InsertedCursor;
    DEALLOCATE InsertedCursor;
END;
```

Wynik:

The screenshot shows the SQL Server Enterprise Manager interface. The left pane displays the 'Database Explorer' with the 'Orders' table selected. The center pane shows the execution of an INSERT statement into the 'Orders' table. The right pane shows the 'Files' pane. The bottom pane shows the 'Services' pane with a list of services. The 'Output' window displays the results of the execution, showing a table with columns 'OrderID' and 'ParticipantsID'.

OrderID	ParticipantsID
14	8
15	9
16	10
17	15
18	16
19	17

4. Trigger na datę płatności

Trigger, który ustawia datę płatności na 7 dni przed datą wycieczki (Mateusz)

```
CREATE TRIGGER trg_AfterInsert_Orders
ON Orders
AFTER INSERT
AS
BEGIN
    UPDATE O
    SET O.PaymentDue = DATEADD(day, -7, T.TourDateBegins)
    FROM Orders O
    INNER JOIN inserted i ON O.OrderID = i.OrderID
    INNER JOIN Tours T ON O.TourID = T.TourID;
END;
```

Wynik:

The screenshot shows the SQL Server Enterprise Manager interface. The left pane displays the 'Database Explorer' with the 'Orders' table selected. The right pane shows the 'console_12' window with the following SQL script and error messages:

```
1 INSERT INTO Orders (OrderID, ClientID, OrderDate, OrderStatusID, TourID, Discount, PaymentDue, EmployeeID)
2 VALUES (16, 1, '2023-01-01', 1, 1, 0.00, '2023-01-01', 1);
```

Error messages:

- [2024-06-11 10:55:16] [23000][515] Line 1: Cannot insert the value NULL into column 'PaymentDue', table 'u_wirkijow.dbo.Orders'. The statement has been terminated.
- [2024-06-11 10:55:16] [S0000][3621] The statement has been terminated.
- [2024-06-11 10:55:51] [23000][2627] Line 1: Violation of PRIMARY KEY constraint 'Orders_pk'. Cannot insert duplicate key (16, 1) in table 'u_wirkijow.dbo.Orders'. The statement has been terminated.
- [2024-06-11 10:55:51] [S0000][3621] The statement has been terminated.
- [2024-06-11 10:55:59] INSERT INTO Orders (OrderID, ClientID, OrderDate, OrderStatusID, TourID, Discount, PaymentDue, EmployeeID) VALUES (16, 1, '2023-01-01', 1, 1, 0.00, '2023-01-01', 1) 2 rows affected in 23 ms

The screenshot shows the SQL Server Enterprise Manager interface. The left pane displays the 'Database Explorer' with the 'Orders' table selected. The right pane shows the 'console_12' window with the following SQL script and error messages:

```
1 WHERE
2 ORDER BY
3 ClientID
4 OrderDate
5 OrderStatusID
6 TourID
7 Discount
8 PaymentDue
9 EmployeeID
```

Table data:

ClientID	OrderDate	OrderStatusID	TourID	Discount	PaymentDue	EmployeeID
4	2023-04-25	4	4	25.0000	2023-03-25	
5	2023-05-30	5	1	30.0000	2023-06-30	
6	2023-06-10	6	2	35.0000	2023-07-10	
7	2023-07-15	1	3	40.0000	2023-08-15	
8	2023-08-20	2	4	45.0000	2023-09-20	
9	2023-09-25	3	1	50.0000	2023-10-25	
10	2023-10-30	4	2	55.0000	2023-11-30	
11	2023-11-01	5	3	60.0000	2023-12-01	
12	2023-12-15	6	4	65.0000	2024-01-15	
13	2024-05-27	1	1	10.0000	2024-06-03	
14	2024-06-09	1	1	10.0000	2024-06-10	
15	2023-01-01	2	1	0.0000	2024-06-11	
16	2023-01-01	1	1	0.0000	2022-12-25	

5. Trigger na sprawdzanie e-mail uczestników

Uruchamiany jest przed aktualizacją rekordu w tabeli 'Clients' - sprawdzając, czy adres e-mail jest unikalny (Mateusz)

```

CREATE TRIGGER trg_BeforeUpdate_Clients
ON Clients
INSTEAD OF UPDATE
AS
BEGIN
    IF EXISTS (SELECT 1 FROM Clients c, inserted i WHERE c.Email =
i.Email AND c.ClientID <> i.ClientID)
    BEGIN
        RAISERROR('Email must be unique.', 16, 1);
        ROLLBACK TRANSACTION;
    END
    ELSE
    BEGIN
        UPDATE Clients
        SET FirstName = i.FirstName,
            LastName = i.LastName,
            Email = i.Email,
            PhoneNumber = i.PhoneNumber,
            BirthDate = i.BirthDate,
            PostalCode = i.PostalCode,
            Address = i.Address,
            PlaceID = i.PlaceID
        FROM inserted i
        WHERE Clients.ClientID = i.ClientID;
    END;
END;

```

6. Triger wywołany po nowej wpłacie

Po wpłynięciu nowej wpłaty trigger ten uruchamia procedurę

CheckAndUpdateOrderStatus, która sprawdza czy już wpłynęła pełna kwota i jeśli tak to zmienia jej status z pending na confirmed. (Patryk)

```

CREATE TRIGGER trg_AfterInsert_Payments
ON Payments
AFTER INSERT
AS
BEGIN
    DECLARE @OrderID INT;

    SELECT @OrderID = OrderID
    FROM inserted;

    EXEC CheckAndUpdateOrderStatus @OrderID;
END;

```

Przed wpłatą:

	OrderID	OrderStatusID	OrderStatus
1	1	1	Pending

Po wpłacie:

```
1 SELECT OrderID, Orders.OrderStatusID, OrderStatus.OrderStatus FROM Orders
2 Inner JOIN OrderStatus ON Orders.OrderStatusID = OrderStatus.OrderStatusID
3
4 SELECT * FROM OrderFinalPrice
5
6 SELECT * FROM Payments
7
8 INSERT INTO Payments (PaymentID, OrderID, Amount, Date)
```

Results Messages

	OrderID	OrderStatusID	OrderStatus
1	1	2	Confirmed

7. Trigger który przypisuje pracownika do domyślnego menedżera

Jeśli kolumna ReportsTo jest pusta podczas tworzenia nowego pracownika to będzie on przypisany do domyślnego menedżera. (Patrik)

```
CREATE TRIGGER trg_AfterInsert_Employees
ON Employees
AFTER INSERT
AS
BEGIN
    UPDATE E
    SET E.ReportsTo = (SELECT EmployeeID FROM Employees WHERE PositionID
= 2)
    FROM Employees E
    INNER JOIN inserted i ON E.EmployeeID = i.EmployeeID
    WHERE E.ReportsTo IS NULL;
END;
```

Komenda:

```
INSERT INTO Employees (EmployeeID, FirstName, LastName, BirthDate, HireDate, Address,
PostalCode, PhoneNumber, Email, Extension, Photo, ReportsTo, PositionID, PlaceID)
VALUES (12, 'Sophia', 'Green', '1992-11-22', '2024-06-15', 'Pine St 321', '00-012',
'987654321', 'sophia.green@example.com', '5678', 0x00, NULL, 3, 5);
```

Wynik:

	EmployeeID	FirstName	LastName	ReportsTo
11	11	Holt	Raymond	NULL
12	12	Sophia	Green	2

8. Trigger pomagający usunąć pracownika i jego powiązania

Uruchamia się gdy usuniemy pracownika, usuwając również jego powiązania tego pracownika w tabelach wycieczki, usługi i atrakcje. (Patrik)

```
CREATE TRIGGER trg_InsteadOfDelete_Employees
ON Employees
INSTEAD OF DELETE
AS
BEGIN
    UPDATE Orders
    SET EmployeeID = 1
    WHERE EmployeeID IN (SELECT EmployeeID FROM deleted);

    DELETE FROM Employee_Tour
    WHERE EmployeeID IN (SELECT EmployeeID FROM deleted);

    DELETE FROM Employee_Service
    WHERE EmployeeID IN (SELECT EmployeeID FROM deleted);

    DELETE FROM Employee_Attraction
    WHERE EmployeeID IN (SELECT EmployeeID FROM deleted);

    DELETE FROM Employees
    WHERE EmployeeID IN (SELECT EmployeeID FROM deleted);
END;
```

Komenda

```
DELETE FROM Employees
WHERE EmployeeID = @EmployeeID;
```

9. Trigger sprawdzający unikalność nazwy atrakcji

Uruchamia się przed dodaniem nowej atrakcji sprawdzając czy nazwa jest unikalna (Mateusz)

```
CREATE TRIGGER trg_BeforeInsert_Atractions
ON Atractions
INSTEAD OF INSERT
AS
BEGIN
    IF EXISTS (SELECT 1 FROM Atractions a, inserted i WHERE a.Name =
i.Name)
    BEGIN
        RAISERROR('Attraction name must be unique.', 16, 1);
        ROLLBACK TRANSACTION;
    END
    ELSE
    BEGIN
        INSERT INTO Atractions (AttractionID, Name, Price, Description,
MaxAtendents, TourID)
        SELECT AttractionID, Name, Price, Description, MaxAtendents,
TourID
        FROM inserted;
    END;
END;
```

10. Trigger sprawdzający unikatowość nazwy wycieczki w danej wycieczce (Mieszko)

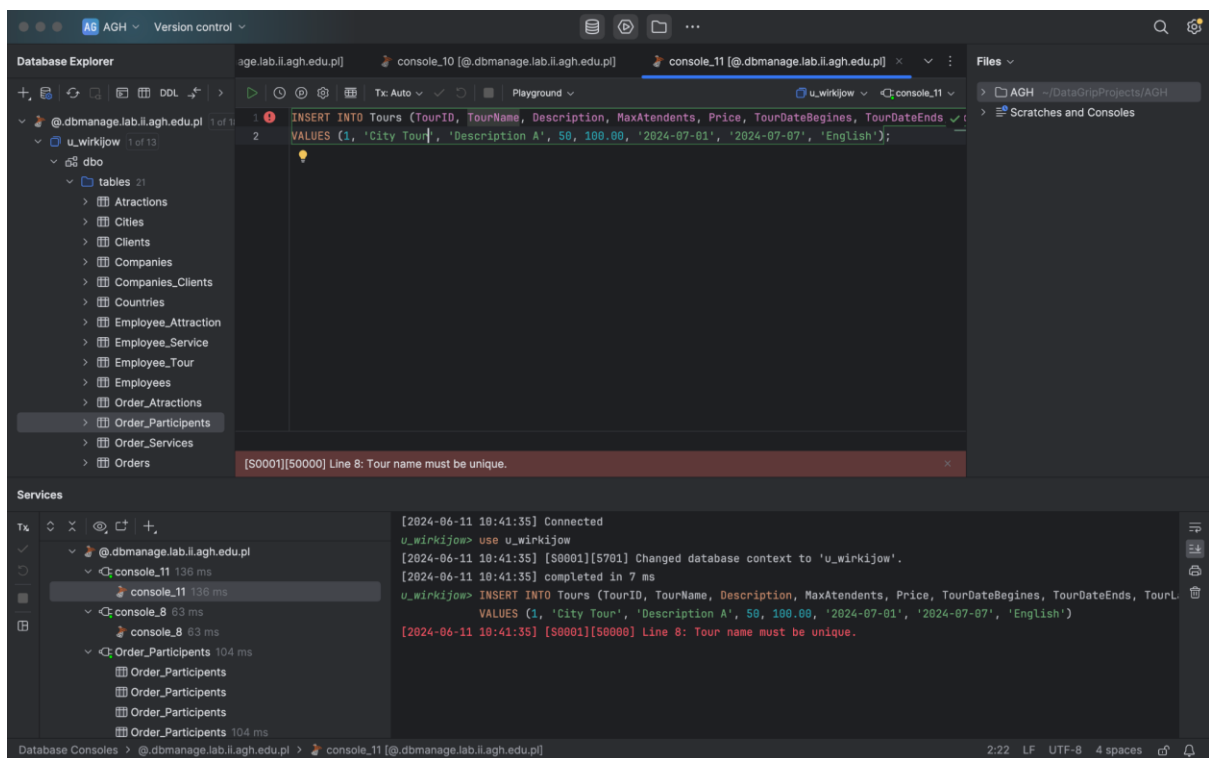
```
CREATE TRIGGER trg_BeforeInsert_Tours
ON Tours
INSTEAD OF INSERT
AS
BEGIN
    IF EXISTS (SELECT 1 FROM Tours a, inserted i WHERE a.TourName =
i.TourName)
    BEGIN
        RAISERROR('Tour name must be unique.', 16, 1);
        ROLLBACK TRANSACTION;
    END
END
```

```

ELSE
BEGIN
    INSERT INTO Tours (TourID, Tourname, Description, MaxAtendents,
Price, TourDateBegin, TourDateEnds, TourLanguage)
        SELECT TourID, Tourname, Description, MaxAtendents, Price,
TourDateBegin, TourDateEnds, TourLanguage
        FROM inserted;
END;
END;

```

Wynik



11. Trigger sprawdzający unikatowość nazwy firmy zamawiającej wycieczkę (Mieszko)

```

CREATE TRIGGER trg_BeforeInsert_Companies
ON Companies
INSTEAD OF INSERT
AS
BEGIN
    IF EXISTS (SELECT 1 FROM Companies a, inserted i WHERE a.Name =
i.Name)
    BEGIN
        RAISERROR('Companies name must be unique.', 16, 1);
        ROLLBACK TRANSACTION;
    END
END

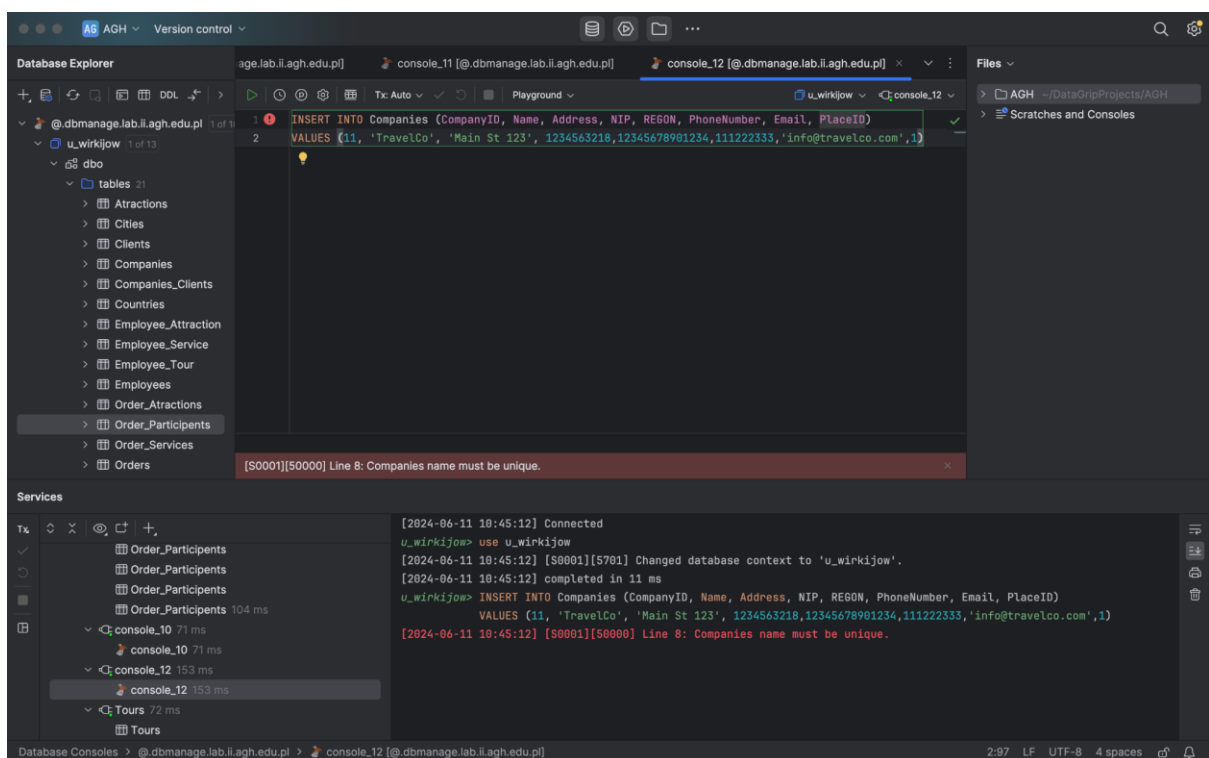
```

```

END
ELSE
BEGIN
    INSERT INTO Companies (CompanyID, Name, Address, NIP, REGON,
    PhoneNumber, Email, PlaceID)
    SELECT CompanyID, Name, Address, NIP, REGON, PhoneNumber, Email,
    PlaceID
    FROM inserted;
END;
END;

```

Wynik



12. Trigger sprawdzający czy można dokonać zmiany w zamówieniu.

Jeśli wycieczka rozpoczyna się za mniej niż 7 dni nie można dokonać żadnych zmian w zamówieniu.

```

CREATE TRIGGER trg_BeforeUpdate_Tours
ON Tours
INSTEAD OF Update
AS
BEGIN
    IF EXISTS

```

```

        SELECT 1
        FROM inserted
        JOIN Tours ON inserted.TourID = Tours.TourID
        WHERE DATEDIFF(day, GETDATE(), Tours.TourDateBegin) < 7
    BEGIN
        RAISERROR('You can not change Tour 7 before Tours Start.', 16,
1);
        ROLLBACK TRANSACTION;
    END
    ELSE
    BEGIN
        UPDATE Tours
        SET Tourname = inserted.Tourname,
        Description = inserted.Description,
        MaxAtendents = inserted.MaxAtendents,
        Price = inserted.Price,
        TourDateBegin = inserted.TourDateBegin,
        TourDateEnds = inserted.TourDateEnds,
        TourLanguage = inserted.TourLanguage
        FROM inserted
        WHERE Tours.TourID = inserted.TourID;
    END;
END;

```

Funkcje

1. Wyświetl wycieczki, atrakcje i serwisy dla danego uczestnika(Karol)

```

CREATE FUNCTION GetParticipantDetails(@ParticipantID INT)
RETURNS @Result TABLE (
    Type NVARCHAR(50),
    ID INT,
    Name NVARCHAR(100),
    Description NVARCHAR(MAX),
    Price MONEY

```

```

)
AS
BEGIN
    INSERT INTO @Result (Type, ID, Name, Description, Price)
    SELECT 'Tour', t.TourID, t.TourName, t.Description, t.Price
    FROM Participants p
    JOIN Order_Participants op ON p.ParticipantID = op.ParticipantID
    JOIN Orders o ON op.OrderID = o.OrderID
    JOIN Tours t ON o.TourID = t.TourID
    WHERE p.ParticipantID = @ParticipantID;

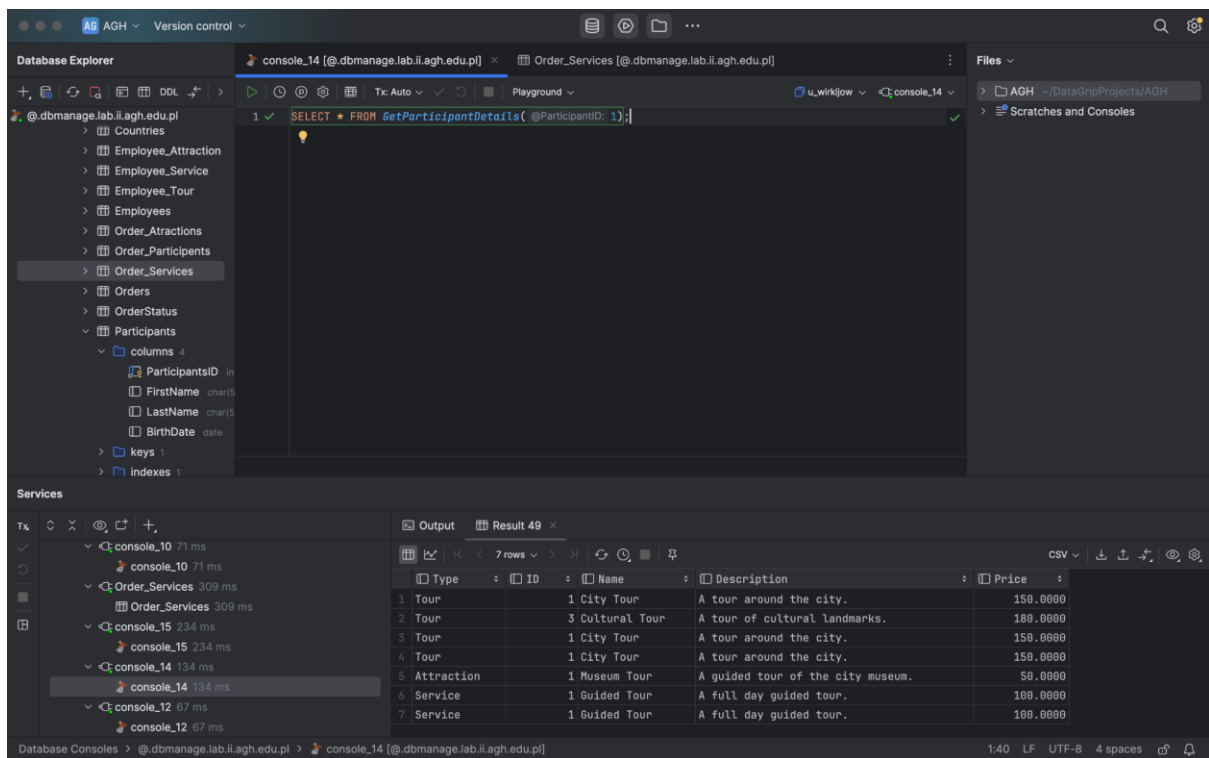
    INSERT INTO @Result (Type, ID, Name, Description, Price)
    SELECT 'Attraction', a.AttractionID, a.Name, a.Description, a.Price
    FROM Participants p
    JOIN Order_Attractions oa ON p.ParticipantID = oa.ParticipantID
    JOIN Attractions a ON oa.AttractionID = a.AttractionID
    WHERE p.ParticipantID = @ParticipantID;

    INSERT INTO @Result (Type, ID, Name, Description, Price)
    SELECT 'Service', s.ServiceID, s.Name, s.Description, s.Price
    FROM Participants p
    JOIN Order_Services os ON p.ParticipantID = os.ParticipantID
    JOIN Services s ON os.ServiceID = s.ServiceID
    WHERE p.ParticipantID = @ParticipantID;

    RETURN;
END;

```

Wynik:



Type	ID	Name	Description	Price
Tour	1	City Tour	A tour around the city.	150.0000
Tour	3	Cultural Tour	A tour of cultural landmarks.	180.0000
Tour	1	City Tour	A tour around the city.	150.0000
Tour	1	City Tour	A tour around the city.	150.0000
Attraction	1	Museum Tour	A guided tour of the city museum.	50.0000
Service	1	Guided Tour	A full day guided tour.	100.0000
Service	1	Guided Tour	A full day guided tour.	100.0000

2. Wyświetl wszystkie atrakcje dla danej wycieczki(Karol)

```
CREATE FUNCTION GetAttractionsForTour(@TourID INT)
RETURNS TABLE
AS
RETURN (
    SELECT a.AttractionID, a.Name, a.Description, a.Price
    FROM Tours t
    JOIN Attractions a ON t.TourID = a.TourID
    WHERE t.TourID = @TourID
);
```

Wynik:

The screenshot shows the SQL Server Enterprise Manager interface. The left pane displays the 'Database Explorer' for the 'dbmanage.lab.ii.agh.edu.pl' database, with the 'Order_Services' database selected. The central pane shows a query window with the following SQL statement: `SELECT * FROM GetAttractionsForTour (@TourID: 1)`. The right pane shows the 'Files' section. The bottom pane displays the 'Services' section, listing various services and their execution times. The 'Output' pane shows the result of the query, which is a table with 4 columns: 'AttractionID', 'Name', 'Description', and 'Price'. The table contains 2 rows of data.

AttractionID	Name	Description	Price
1	Museum Tour	A guided tour of the city museum.	50.0000
2	Art Gallery Visit	A visit to the modern art gallery.	35.0000

Uprawnienia

Administrator

- Generowanie raportów
- Zarządzanie wycieczkami
- Zarządzanie aktywnościami dodatkowymi

```
CREATE ROLE Administrator;  
GRANT SELECT ON Reports TO Administrator;  
GRANT INSERT, UPDATE, DELETE ON Trips TO Administrator;  
GRANT INSERT, UPDATE, DELETE ON Activities TO Administrator;
```

Sprzedawca/Reklamodawca

- Wykonywanie rezerwacji wycieczki
- Wyświetlanie wszystkich wycieczek
- Opinie klienckie

```
CREATE ROLE Salesperson_Advertiser;  
GRANT INSERT ON Reservations TO Salesperson_Advertiser;
```



```
GRANT SELECT ON Trips TO Salesperson_Advertiser;  
GRANT INSERT, UPDATE, SELECT ON Reviews TO Salesperson_Advertiser;
```

Kierownik wycieczki

- Podanie osób biorących udział w wycieczce
- Rezerwacja dodatkowych usług i atrakcji do danej wycieczki
- Modyfikacja danej wycieczki która została zarezerwowana/opłacona
- Anulowanie rezerwacji/atrakcji/dodatkowych usług
- Opinie klienckie
- Transport

```
CREATE ROLE Tour_Manager;  
GRANT INSERT ON Participants TO Tour_Manager;  
GRANT INSERT ON AdditionalServices TO Tour_Manager;  
GRANT UPDATE ON Reservations TO Tour_Manager;  
GRANT DELETE ON Reservations TO Tour_Manager;  
GRANT INSERT, UPDATE, SELECT ON Transport TO Tour_Manager;  
GRANT INSERT, UPDATE, SELECT ON Reviews TO Tour_Manager;
```

Przewodnik wycieczki

- Podanie osób biorących udział w wycieczce

```
CREATE ROLE Tour_Guide;  
GRANT INSERT ON Participants TO Tour_Guide;
```

Prowadzący aktywność dodatkową

- Podanie osób biorących udział w aktywności dodatkowej

```
CREATE ROLE Tour_Guide;  
GRANT INSERT ON Participants TO Tour_Guide;
```

Uczestnik

- Wykonywanie rezerwacji wycieczki
- Dokonanie opłaty za wycieczkę
- Rezerwacja dodatkowych usług i atrakcji do danej wycieczki
- Wyświetlanie wszystkich wycieczek
- Potwierdzenie rezerwacji

- Pobieranie Faktur i paragonów z dokonanych transakcji

```
CREATE ROLE Participant;  
GRANT INSERT ON Reservations TO Participant;  
GRANT UPDATE ON Payments TO Participant;  
GRANT INSERT ON AdditionalServices TO Participant;  
GRANT SELECT ON Trips TO Participant;  
GRANT UPDATE ON Reservations TO Participant;  
GRANT SELECT ON Invoices TO Participant;
```

Uczestnik (firma)

- Takie same uprawnienia jak uczestnik indywidualny

```
CREATE ROLE Corporate_Participant;  
GRANT INSERT ON Reservations TO Corporate_Participant;  
GRANT UPDATE ON Payments TO Corporate_Participant;  
GRANT INSERT ON AdditionalServices TO Corporate_Participant;  
GRANT SELECT ON Trips TO Corporate_Participant;  
GRANT UPDATE ON Reservations TO Corporate_Participant;  
GRANT SELECT ON Invoices TO Corporate_Participant;
```

Potencjalny Klient (niezalogowany)

- Wykonywanie rezerwacji wycieczki
- Podanie osób biorących udział w wycieczce
- Wyświetlanie wszystkich wycieczek

```
CREATE ROLE Potential_Client;  
GRANT INSERT ON Reservations TO Potential_Client;  
GRANT INSERT ON Participants TO Potential_Client;  
GRANT SELECT ON Trips TO Potential_Client;
```