

A G H

**AKADEMIA GÓRNICZO-HUTNICZA
IM. STANISŁAWA STASZICA
W KRAKOWIE**

**Laboratorium Podstaw Baz Danych
Dokumentacja Systemu zarządzania
konferencjami**

Miłosz Mandowski
Michał Śledź

Spis treści

- Ogólne informacje
- Schemat bazy danych
- Specyfikacja wymagań
- Tabele
 - Clients
 - Conferences
 - DaysReservations
 - ParticipantReservations
 - Participants
 - ParticipantWorkshops
 - Payments
 - PriceList
 - Workshops
 - WorkshopsReservations
- Kod tworzący klucze obce
- Kod usuwający wszystkie tabele
- Triggery
 - T_CancelAllDaysReservations
 - T_NoFreePlacesForAnyConferenceDay
 - T_ControlClientSurnameAndIsPrivateStatus
 - T_ControlUpdatingPlacesForConference
 - T_CancelAllParticipantConferenceDayReservations
 - T_CancelAllWorkshopsReservations
 - T_NoPlacesForConferenceDay
 - T_CancelAllParticipantWorkshopsReservations1
 - T_CheckIfParticipantCanBeAdded
 - T_CheckPriceListInsert
 - T_ControlUpdatingPlacesForWorkshop
 - T_CheckIfWorkshopDayBelongsToConferenceDay
 - T_ControlPlacesForWorkshop
 - T_CancelAllParticipantWorkshopsReservations2
 - T_ControlFreePlacesReservedByClientForConferenceDay
 - T_ControlStudentsCardFieldsFilling
 - T_ControlUpdatingPlacesForWorkshop
 - T_DeleteFineAssesdAfterCancelingConferenceReservation
 - T_CountFineAfterWorkhopReservationOrUpdate
 - T_CountFineAfterConferenceDayReservationOrUpdate
 - T_CreatePaymentField
- Procedury
 - P_AddClient
 - P_AddConference
 - P_AddParticipant

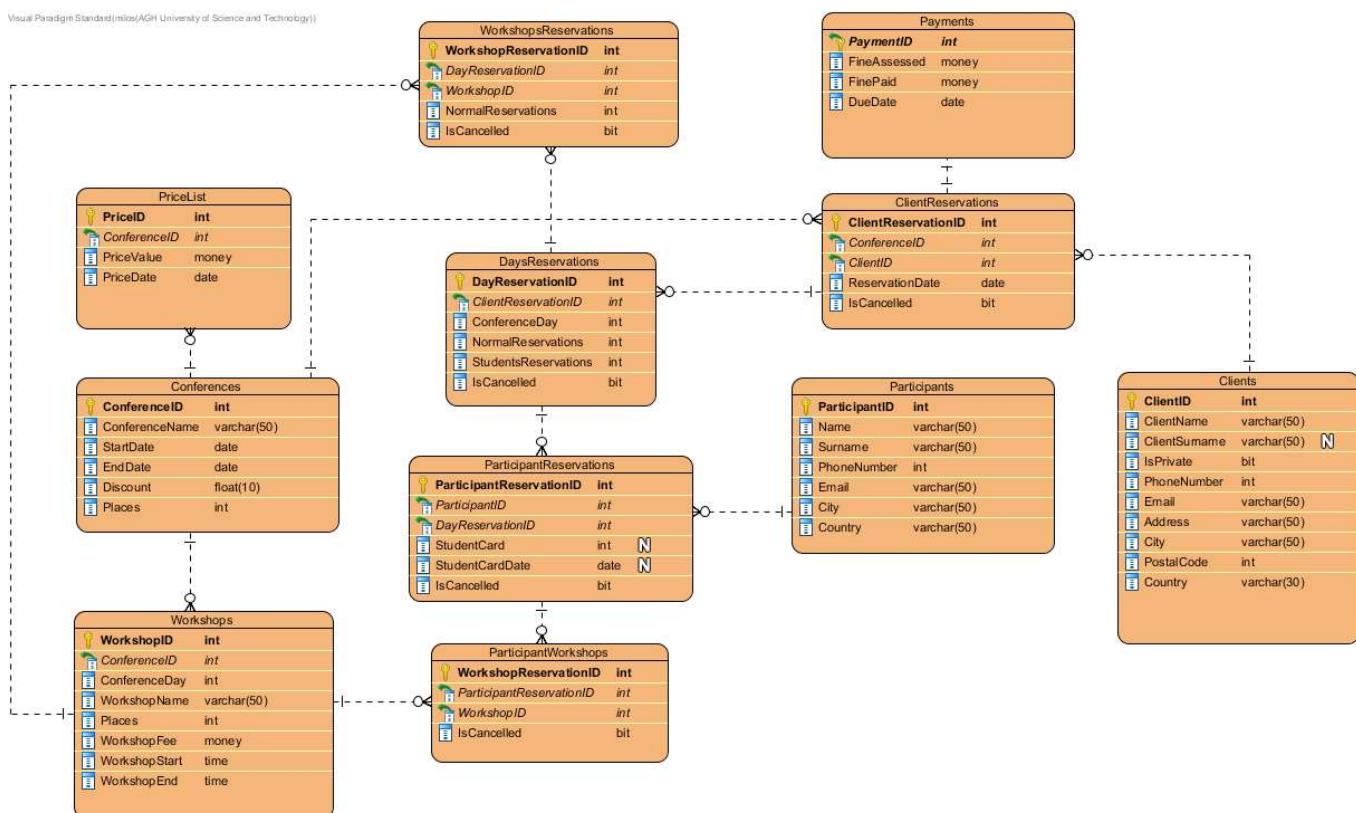
- P_AddParticipantForConferenceDay
 - P_AddParticipantForWorkshop
 - P_AddPriceToConferencePriceList
 - P_AddReservationForConference
 - P_AddReservationForConferenceDay
 - P_AddReservationForWorkshop
 - P_AddWorkshop
 - P_CancelConferenceReservation
 - P_CancelDayReservation
 - P_CancelParticipantReservation
 - P_CancelParticipantWorkshopReservation
 - P_CancelUnpaidReservation
 - P_CancelWorkshopReservation
 - P_ChangeConferenceDetails
 - P_ChangeDayReservationPlaces
 - P_ChangeWorkshopDetails
 - P_ChangeWorkshopReservationPlaces
 - P_CheckCurrentPayment
 - P_CountFine
 - P_DeletePriceFromConferencePriceList
- Funkcje
 - F_AllPaymentsByClientID
 - F_ClientReservationsHistory
 - F_ClientsWithUnusedPlaces
 - F_ConferenceParticipants
 - F_CreatePeopleIdentifiers
 - F_FreeAndReservedPlacesForConference
 - F_FreeAndReservedPlacesForWorkshop
 - F_NonregulatedPaymentsByClientID
 - F_ParticipantsListForConferenceDay
 - F_ParticipantsListForWorkshop
 - F_RegulatedPaymentsByClientID
 - F_ShowPrices
 - F_ShowWorkshops
 - F_GetCurrentPrice
- Widoki
 - V_MostFrequentClients
 - V_MostProfitableClients
 - V_UnpayedCancelledReservations
 - V_UnpayedNotCancelledReservations
 - V_OverPayedReservations
 - V_PayedReservations
 - V_CancelledConferencesReservations
 - V_ClientsList
- Indeksy
- Role

- Administrator
- Pracownik
- Klient
- Uczestnik
- Generator danych

Ogólne informacje

Firma organizuje konferencje, które mogą być jedno- lub kilkudniowe. Klienci powinni móc rejestrować się na konferencje za pomocą systemu www. Klientami mogą być zarówno indywidualne osoby jak i firmy, natomiast uczestnikami konferencji są osoby (firma nie musi podawać od razu przy rejestracji listy uczestników - może zarezerwować odpowiednią ilość miejsc na określone dni oraz na warsztaty, natomiast na 2 tygodnie przed rozpoczęciem musi te dane uzupełnić - a jeśli sama nie uzupełni do tego czasu, to pracownicy dzwonią do firmy i ustalają takie informacje). Każdy uczestnik konferencji otrzymuje identyfikator imienny (+ ew. informacja o firmie na nim). Dla konferencji kilkudniowych, uczestnicy mogą rejestrować się na dowolne z tych dni.

Schemat bazy danych



Specyfikacja wymagań

Podczas analizowania wymagań systemu natknęliśmy się na kilka brzegowych sytuacji. Poniżej przedstawiamy je wraz z ich rozwiązaniem.

- Dodawani klinci mogą być prywatni lub nie, odpowiada za to pole `isPrivate` oraz brak nazwiska
- Klienci są unikalni, rozróżniani za pomocą adresu email
- Klient może zrobić jedną rezerwację na jedną konferencję, gdyby potrzebował więcej miejsc, może po prostu zwiększyć ich ilość, lub całkowicie anulować rezerwacje i zrobić ją na nowo
- Opłaty klienta są wyliczane automatycznie w momencie zrobienia rezerwacji, lub jakiekolwiek jej aktualizacji
- Codziennie uruchamiana jest procedura sprawdzająca, czy klienci zapłacili w czasie, jeżeli nie, ich rezerwacja jest anulowana
- Anulowanie rezerwacji konferencji, anuluje rezerwacje dni, a anulowanie rezerwacji dni, rezerwacje warsztatów
- Przy robieniu rezerwacji na dzień, monitorowana jest ilość wolnych miejsc na ten dzień, podobnie dla rezerwacji warsztatu
- Przy wpisywaniu konferencji sprawdzamy, czy data końca jest po dacie początku
- Dla każdej konferencji podana jest cena zależna od daty, przy czym ze wzrostem daty następuje wzrost ceny
- Przy dodawaniu warsztatów, na odpowiedni dzień, sprawdzane jest, czy konferencja ma taki dzień
- Przy rejestracji uczestnika na konferencje, sprawdzane jest, czy klient zrobił odpowiednią rezerwację
- Przy wpisie na warsztaty sprawdzane jest, czy istnieje rezerwacja na ten warsztat po stronie klienta
- Przy rezerwacji klienta na warsztat, kontrolowana jest maksymalna liczba miejsc

Tabele

Clients - tabela zawierająca informacje o klientach. Zawiera pola

- **ClientID** - identyfikator klienta, unikatowy, zaczyna się od 1, inkrementowany o 1
- **ClientName** - nazwa klienta. W przypadku firm nazwa firmy, w przypadku osoby prywatnej imię
- **ClientSurname** - nazwisko klienta. W przypadku firmy pozostaje wartość `NULL`, w przypadku osoby prywatnej nazwisko
- **IsPrivate** - informacja czy klientem jest firma czy osoba prywatna. Typ bitowy.
- **PhoneNumber** - numer telefonu. Musi posiadać 9 cyfr, a pierwsza nie może być zerem
- **Email** - adres email klienta
- **Address** - adres klienta
- **City** - miasto klienta
- **PostalCode** - kod pocztowy
- **Country** - państwo klienta

```

CREATE TABLE Clients (
    ClientID int IDENTITY(1,1) NOT NULL,
    ClientName varchar(50) NOT NULL,
    ClientSurname varchar(50) NULL,
    IsPrivate bit NOT NULL DEFAULT 0,
    PhoneNumber int NOT NULL CHECK (PhoneNumber like '[1-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]'),
    Email varchar(50) NOT NULL,
    Address varchar(50) NOT NULL,
    City varchar(50) NOT NULL,
    PostalCode int NOT NULL,
    Country varchar(30) NOT NULL,
    PRIMARY KEY (ClientID));

```

Conferences - tabela zawierająca informacje o konferencjach. Zawiera pola:

- ConferenceID - identyfikator konferencji, unikatowy, zaczyna się od 1, inkrementowany o 1
- ConferenceName - temat konferencji
- StartDate - data rozpoczęcia konferencji w formacie yyyy-mm-dd
- EndDate - data zakończenia konferencji w formacie yyyy-mm-dd
- Discount - zniżka dla studentów. Domyślnie wartość ustawiona na 0. Musi być mniejsza lub równa 1.

```

CREATE TABLE Conferences (
    ConferenceID int IDENTITY(1,1) NOT NULL,
    ConferenceName varchar(50) NOT NULL,
    StartDate date NOT NULL,
    EndDate date NOT NULL,
    Discount float(4) NOT NULL DEFAULT 0 CHECK (Discount <= 1),
    CHECK (StartDate <= EndDate),
    PRIMARY KEY (ConferenceID));

```

ClientReservations - przechowuje informacje o rezerwacji klienta na daną konferencję. Zawiera pola:

- ClientReservationID - identyfikator rezerwacji, unikatowy, zaczyna się od 1, inkrementowany o 1
- ConferenceID - identyfikator konferencji
- ClientID - identyfikator klienta

- **ReservationDate** - data dokonania rezerwacji

```
CREATE TABLE ClientReservations (
    ClientReservationID int IDENTITY(1,1) NOT NULL,
    ConferenceID int NOT NULL,
    ClientID int NOT NULL,
    ReservationDate date NOT NULL DEFAULT Convert(date, getdate()),
    PRIMARY KEY (ClientReservationID));
```

DaysReservations - przechowuje informacje o rezerwacjach klientów na konkretne dni konferencji.

Zawiera pola:

- **DayReservationID** - identyfikator rezerwacji na dany dzień, unikatowy, zaczyna się od 1, inkrementowany o 1
- **ClientReservationID** - identyfikator klienta
- **ConferenceDay** - nr dnia, na który dokonano rezerwacji
- **NormalReservations** - ilość zarezerwowanych normalnych miejsc, musi być większa od zera
- **StudentsReservations** - ilość zarezerwowanych studenckich miejsc, domyślnie równa zero

```
CREATE TABLE DaysReservations (
    DayReservationID int IDENTITY(1,1) NOT NULL,
    ClientReservationID int NOT NULL,
    ConferenceDay int NOT NULL,
    NormalReservations int NOT NULL CHECK (NormalReservations > 0),
    StudentsReservations int NOT NULL DEFAULT 0,
    PRIMARY KEY (DayReservationID));
```

ParticipantReservations - przechowuje informacje o zapisach uczestników na dany dzień konferencji.

Zawiera pola:

- **ParticipantReservationID** - identyfikator rezerwacji uczestnika, unikatowy, zaczyna się od 1, inkrementowany o 1
- **ParticipantID** - identyfikator uczestnika
- **DayReservationID** - identyfikator rezerwacji klienta na dany dzień
- **StudentCard** - nr legitymacji studenckiej, równy **null** jeżeli uczestnik nie jest studentem
- **StudentCardDate** - ważność legitymacji studenckiej, równa **null** jeżeli uczestnik nie jest studentem

```
CREATE TABLE ParticipantReservations (
    ParticipantReservationID int IDENTITY(1,1) NOT NULL,
```

```
ParticipantID int NOT NULL,  
DayReservationID int NOT NULL,  
StudentCard int NULL,  
StudentCardDate date NULL,  
PRIMARY KEY (ParticipantReservationID));
```

Participants - przechowuje informacje o uczestnikach konferencji. Zawiera pola:

- ParticipantID - identyfikator uczestnika, unikatowy, zaczyna się od 1, inkrementowany o 1
- Name - imię uczestnika
- Surname - nazwisko uczestnika
- PhoneNumber - nr telefonu, musi posiadać 9 cyfr, pierwsza musi być różna od zera
- Email - adres email
- City - miasto
- Country - państwo
- DiscountGranted - informuje czy uczestnikowi przysługuje zniżka studencka, wartość bitowa, domyślnie nie przysługuje potrzebne nam????

```
CREATE TABLE Participants (  
    ParticipantID int IDENTITY(1,1) NOT NULL,  
    Name varchar(50) NOT NULL,  
    Surname varchar(50) NOT NULL,  
    PhoneNumber int NOT NULL CHECK (PhoneNumber like '[1-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]'),  
    Email varchar(50) NOT NULL,  
    City varchar(50) NOT NULL,  
    Country varchar(50) NOT NULL,  
    DiscountGranted bit NOT NULL DEFAULT 0,  
    PRIMARY KEY (ParticipantID));
```

ParticipantWorkshops - przechowuje informacje o zapisach uczestników na warsztaty. Zawiera pola:

- WorkshopReservationID - identyfikator rezerwacji na warsztat, unikatowy, zaczyna się od 1, inkrementowany o 1
- ParticipantReservationID - identyfikator rezerwacji uczestnika na dany dzień konferencji
- WorkshopID - identyfikator warsztatu

```
CREATE TABLE ParticipantWorkshops (  
    WorkshopReservationID int IDENTITY(1,1) NOT NULL,  
    ParticipantReservationID int NOT NULL,
```

```
WorkshopID int NOT NULL,  
PRIMARY KEY (WorkshopReservationID));
```

Payments - przechowuje informacje o opłatach nałożonych na klientów, za dokonane rezerwacje miejsc na konferencje i warsztaty. Zawiera pola:

- PaymentID - identyfikator opłaty, unikatowy zaczyna się od 1, inkrementowany o 1
- FineAssessed - należna opłata za rezerwacje miejsc na konferencję i warsztaty
- FinePaid - kwota zapłacona do tej pory
- DueDate - czas, do którego należy dokonać opłaty

```
CREATE TABLE Payments (  
PaymentID int IDENTITY(1,1) NOT NULL,  
FineAssessed money NOT NULL,  
FinePaid money NOT NULL DEFAULT 0,  
DueDate date NOT NULL,  
PRIMARY KEY (PaymentID));
```

PriceList - przechowuje informacje o cenach za rezerwacje na konferencję w zależności od daty dokonania rezerwacji. Zawiera pola:

- PriceID - identyfikator opłaty
- ConferenceID - identyfikator konferencji
- PriceValue - cena za rezerwację
- PriceDate - data, do której obowiązuje dana cena

```
CREATE TABLE PriceList (  
PriceID int IDENTITY(1,1) NOT NULL,  
ConferenceID int NOT NULL,  
PriceValue money NOT NULL,  
PriceDate date NOT NULL,  
PRIMARY KEY (PriceID));
```

Workshops - przechowuje informacje o warsztatach. Zawiera pola:

- WorkshopID - identyfikator warsztatu, unikatowy, zaczyna się od 1, inkrementowany o 1
- ConferenceID - identyfikator konferencji
- ConferenceDay - nr dnia konferencji, na który przypada warsztat

- **WorkshopName** - temat/nazwa warsztatu
- **Places** - ilość dostępnych miejsc
- **WorkshopFee** - wysokość opłaty należnej za wstęp na warsztat
- **WorkshopStart** - czas rozpoczęcia warsztatu
- **WorkshopEnd** - czas zakończenia warsztatu

```
CREATE TABLE Workshops (
    WorkshopID int IDENTITY(1,1) NOT NULL,
    ConferenceID int NOT NULL,
    ConferenceDay int NOT NULL,
    WorkshopName varchar(50) NOT NULL,
    Places int NOT NULL CHECK (Places >= 0),
    WorkshopFee money NOT NULL,
    WorkshopStart time NOT NULL,
    WorkshopEnd time NOT NULL,
    CHECK (WorkshopStart < WorkshopEnd),
    PRIMARY KEY (WorkshopID));
```

WorkshopsReservations - przechowuje infomacje o rezerwacjach miejsc na warsztaty przez klientó.

Zawiera pola:

- **WorkshopReservationID** - identyfikator rezerwacji na warsztat, unikatowy, zaczyna się od 1, inkrementowany o 1
- **DayReservationID** - identyfikator rezerwacji klienta na dany dzień
- **WorkshopID** - identyfikator warsztatu
- **NormalReservations** - ilość zarezerwowanych miejsc. Musi być większa od zera

```
CREATE TABLE WorkshopsReservations (
    WorkshopReservationID int IDENTITY(1,1) NOT NULL,
    DayReservationID int NOT NULL,
    WorkshopID int NOT NULL,
    NormalReservations int NOT NULL CHECK (NormalReservations > 0),
    PRIMARY KEY (WorkshopReservationID));
```

Kod tworzący klucze obce

```
-- =====
```

```
-- Creating Keys
```

```
-- =====
```

```
ALTER TABLE WorkshopsReservations
```

```
    ADD CONSTRAINT FKWorkshopsResToDaysRes
```

```
        FOREIGN KEY (DayReservationID) REFERENCES DaysReservations (DayReservationID);
```

```
ALTER TABLE WorkshopsReservations
```

```
    ADD CONSTRAINT FKWorkshopsResToWorkshops
```

```
        FOREIGN KEY (WorkshopID) REFERENCES Workshops (WorkshopID);
```

```
ALTER TABLE ParticipantReservations
```

```
    ADD CONSTRAINT FKParticipantResToDaysRes
```

```
        FOREIGN KEY (DayReservationID) REFERENCES DaysReservations (DayReservationID);
```

```
ALTER TABLE ParticipantReservations
```

```
    ADD CONSTRAINT FKParticipantResToParticipants
```

```
        FOREIGN KEY (ParticipantID) REFERENCES Participants (ParticipantID);
```

```
ALTER TABLE ParticipantWorkshops
```

```
    ADD CONSTRAINT FKParticipantWorksToParticipantRes
```

```
        FOREIGN KEY (ParticipantReservationID) REFERENCES ParticipantReservations (ParticipantReservationID);
```

```
ALTER TABLE ParticipantWorkshops
```

```
    ADD CONSTRAINT FKParticipantWorksToWorkshops
```

```
        FOREIGN KEY (WorkshopID) REFERENCES Workshops (WorkshopID);
```

```
ALTER TABLE ClientReservations
```

```
    ADD CONSTRAINT FKClientResToClients
```

```
        FOREIGN KEY (ClientID) REFERENCES Clients (ClientID);
```

```
ALTER TABLE ClientReservations
```

```
    ADD CONSTRAINT FKClientResToConferences
```

```
        FOREIGN KEY (ConferenceID) REFERENCES Conferences (ConferenceID);
```

```
ALTER TABLE DaysReservations
```

```
    ADD CONSTRAINT FKDaysResToClientRes
```

```
        FOREIGN KEY (ClientReservationID) REFERENCES ClientReservations (ClientReservationID);
```

```
ALTER TABLE Payments
    ADD CONSTRAINT FKPaymentsToClientRes
        FOREIGN KEY (PaymentID) REFERENCES ClientReservations (ClientReservationID);

ALTER TABLE PriceList
    ADD CONSTRAINT FKPriceListToConferences
        FOREIGN KEY (ConferenceID) REFERENCES Conferences (ConferenceID);

ALTER TABLE Workshops
    ADD CONSTRAINT FKWorkshopsToConferences
        FOREIGN KEY (ConferenceID) REFERENCES Conferences (ConferenceID);
```

Kod usuwający wszystkie tabele

```
-- =====
-- Drop code
-- =====

ALTER TABLE WorkshopsReservations
    DROP CONSTRAINT FKWorkshopsResToDaysRes;

ALTER TABLE WorkshopsReservations
    DROP CONSTRAINT FKWorkshopsResToWorkshops;

ALTER TABLE ParticipantReservations
    DROP CONSTRAINT FKParticipantResToDaysRes;

ALTER TABLE ParticipantReservations
    DROP CONSTRAINT FKParticipantResToParticipants;

ALTER TABLE ParticipantWorkshops
    DROP CONSTRAINT FKParticipantWorksToParticipantRes;

ALTER TABLE ParticipantWorkshops
    DROP CONSTRAINT FKParticipantWorksToWorkshops;
```

```
ALTER TABLE ClientReservations
    DROP CONSTRAINT FKClientResToClients;

ALTER TABLE ClientReservations
    DROP CONSTRAINT FKClientResToConferences;

ALTER TABLE DaysReservations
    DROP CONSTRAINT FKDaysResToClientRes;

ALTER TABLE Payments
    DROP CONSTRAINT FKPaymentsToClientRes;

ALTER TABLE PriceList
    DROP CONSTRAINT FKPriceListToConferences;

ALTER TABLE Workshops
    DROP CONSTRAINT FKWorkshopsToConferences;

DROP TABLE ClientReservations;
DROP TABLE Clients;
DROP TABLE Conferences;
DROP TABLE DaysReservations;
DROP TABLE ParticipantReservations;
DROP TABLE Participants;
DROP TABLE ParticipantWorkshops;
DROP TABLE Payments;
DROP TABLE PriceList;
DROP TABLE Workshops;
DROP TABLE WorkshopsReservations;
```

Triggery

T_CancelAllDaysReservations - Anuluje wszystkie rezerwacje na dni konferencji, po anulowaniu rezerwacji na samą konferencję.

```
CREATE TRIGGER T_CancelAllDaysReservations
    ON ClientReservations
```

```
AFTER UPDATE
```

```
AS
```

```
BEGIN
```

```
DECLARE @ClientReservationID      int  
= (SELECT ClientReservationID FROM inserted WHERE IsCancelled = 1)
```

```
UPDATE DaysReservations  
SET IsCancelled = 1  
WHERE ClientReservationID = @ClientReservationID
```

```
END
```

```
GO
```

T_NoFreePlacesForAnyConferenceDay - Blokuje rezerwacje na konferencję jeśli na żaden dzień nie ma już wolnych miejsc.

```
CREATE TRIGGER T_NoFreePlacesForAnyConferenceDay
```

```
ON ClientReservations
```

```
AFTER INSERT
```

```
AS
```

```
BEGIN
```

```
DECLARE @ConferenceID      int  
= (SELECT ConferenceID FROM inserted)
```

```
IF
```

```
(
```

```
    SELECT COUNT(*)  
    FROM F_FreeAndReservedPlacesForConference (@ConferenceID)  
    WHERE FreePlaces > 0
```

```
) = 0
```

```
BEGIN
```

```
    RAISERROR ('Nie ma już wolnych miejsc na żaden dzień tej konferencji.', -1,  
-1)
```

```
    ROLLBACK TRANSACTION
```

```
END
```

END

GO

T_ControlClientSurnameAndIsPrivateStatus - Sprawdza czy podano nazwisko dla klienta prywatnego i czy nie podano nazwiska dla klienta firmowego.

```
CREATE TRIGGER T_ControlClientSurnameAndIsPrivateStatus
    ON Clients
    AFTER INSERT, UPDATE
AS
BEGIN

    DECLARE @IsPrivate      bit
        = (SELECT IsPrivate FROM inserted)

    DECLARE @ClientSurname  varchar(50)
        = (SELECT ClientSurname FROM inserted)

    IF @IsPrivate = 1 AND @ClientSurname IS NULL
    BEGIN
        RAISERROR ('Prywatny klient wymaga podania nazwiska.', -1, -1)
        ROLLBACK TRANSACTION
    END

    IF @IsPrivate = 0 AND @ClientSurname IS NOT NULL
    BEGIN
        RAISERROR ('Dla klienta firmowego nie należy podawać nazwiska.', -1, -1)
        ROLLBACK TRANSACTION
    END
END
GO
```

T_ControlUpdatingPlacesForConference - Blokuje zmniejszenie liczby miejsc na konferencje jeżeli ilość do tej pory zarezerwowanych miejsc jest większa od nowo podanej liczby miejsc.

```

CREATE TRIGGER T_ControlUpdatingPlacesForConference
    ON Conferences
    AFTER UPDATE
AS
BEGIN

    DECLARE @ConferenceID      int
        = ( SELECT ConferenceID FROM inserted)

    DECLARE @NewPlaces      int
        = ( SELECT Places FROM inserted)

    DECLARE @ReservedPlaces      int
        = ( SELECT TOP 1 ReservedPlaces FROM F_FreeAndReservedPlacesForConference (
@ConferenceID) ORDER BY ReservedPlaces DESC)

    IF @NewPlaces < @ReservedPlaces
        BEGIN
            RAISERROR ('Nowa ilosc dostepnych miejsc jest mniejsza od juz zarezerwowane
j.', -1, -1)
            ROLLBACK TRANSACTION
        END
    END
    GO

```

T_CancelAllParticipantConferenceDayReservations - Anuluje wszystkie zapisy uczestników od klienta na dzień konferencji, na który klient anulował swoją rezerwację.

```

CREATE TRIGGER T_CancelAllParticipantConferenceDayReservations
    ON DaysReservations
    AFTER UPDATE
AS
BEGIN

    UPDATE ParticipantReservations
        SET IsCancelled = 1
        WHERE DayReservationID IN

```

```
    (
        SELECT DayReservationID
        FROM inserted
        WHERE IsCancelled = 1
    )

END
GO
```

T_CancelAllWorkshopsReservations - Anuluje wszystkie rezerwacje klienta na warsztaty z danego dnia, jeżeli klient anulował rezerwację na dany dzień.

```
CREATE TRIGGER T_CancelAllWorkshopsReservations
    ON DaysReservations
    AFTER UPDATE
AS
BEGIN

    UPDATE WorkshopsReservations
    SET IsCancelled = 1
    WHERE DayReservationID IN
    (
        SELECT DayReservationID
        FROM inserted
        WHERE IsCancelled = 1
    )

END
GO
```

T_NoPlacesForConferenceDay - Blokuje rezerwacje lub zmiane ilości miejsc na dany dzień konferencji jeżeli nie ma tylu wolnych miejsc ile chce klient.

```
CREATE TRIGGER T_NoPlacesForConferenceDay
    ON DaysReservations
    AFTER INSERT, UPDATE
```

```

AS
BEGIN

DECLARE @ClientReservationID      int
= ( SELECT ClientReservationID FROM inserted)

DECLARE @ConferenceDay      int
= ( SELECT ConferenceDay FROM inserted)

DECLARE @Places      int
= (
    SELECT i.NormalReservations + i.StudentsReservations - d.NormalRese
rvations - d.StudentsReservations
        FROM inserted AS i
        INNER JOIN deleted AS d
            ON i.DayReservationID = d.DayReservationID
    )

DECLARE @ConferenceID      int
= ( SELECT ConferenceID FROM ClientReservations WHERE ClientReservationID =
@ClientReservationID)

DECLARE @FreePlaces      int
= ( SELECT FreePlaces FROM F_FreeAndReservedPlacesForConference (@Conferenc
eID) WHERE ConferenceDay = @ConferenceDay)

IF @Places > @FreePlaces
BEGIN
    RAISERROR ('Nie ma tylu wolnych miejsc na ten dzien konferencji.', -1, -1)
    ROLLBACK TRANSACTION
END

END
GO

```

T_CancelAllParticipantWorkshopsReservations1 - Anuluje wszystkie zapisy uczestnika na warsztaty w danym dniu, jeżeli uczestnik anulował swój zapis na konferencję w tym dniu.

```

CREATE TRIGGER T_CancelAllParticipantWorkshopsReservations1
    ON ParticipantReservations
    AFTER UPDATE
AS
BEGIN

    UPDATE ParticipantWorkshops
    SET IsCancelled = 1
    WHERE ParticipantReservationID IN
    (
        SELECT ParticipantReservationID
        FROM inserted
        WHERE IsCancelled = 1
    )
    AND IsCancelled = 0 -- bo anulowanie moze nastapic po anulowaniu rezerwacji
    na warsztat, albo po anulowaniu rezerwacji na dzien

END
GO

```

T_CheckIfParticipantCanBeAdded - Sprawdza czy można dodać uczestnika na warsztat.

```

CREATE TRIGGER T_CheckIfParticipantCanBeAdded
    ON ParticipantWorkshops
    AFTER INSERT
AS
BEGIN

    DECLARE @ParticipantReservationID INT
        = (SELECT ParticipantReservationID FROM inserted)
    DECLARE @WorkshopID INT
        = (SELECT WorkshopID FROM inserted)
    DECLARE @DayReservationID INT
        = ( SELECT DayReservationID FROM ParticipantReservations
            WHERE ParticipantReservationID = @ParticipantReservationID )

    IF NOT EXISTS (
        SELECT * FROM ParticipantReservations pr

```

```

JOIN DaysReservations dr ON dr.DayReservationID = pr.DayReservationID
JOIN WorkshopsReservations wr ON wr.DayReservationID = dr.DayReservationID
WHERE wr.WorkshopID = @WorkshopID AND pr.ParticipantReservationID = @ParticipantReservationID
AND pr.IsCancelled = 0 AND wr.IsCancelled = 0
)
BEGIN
RAISERROR ('Klient nie zrobił rezerwacji na ten warsztat', -1, -1)
ROLLBACK TRANSACTION
END

IF (
SELECT count(*) FROM DaysReservations dr
JOIN ParticipantReservations pr ON pr.DayReservationID = dr.DayReservationID
JOIN ParticipantWorkshops pw ON pw.ParticipantReservationID = pr.ParticipantReservationID
WHERE dr.DayReservationID = @DayReservationID AND pw.WorkshopID = @WorkshopID
) >= (
SELECT wr.NormalReservations FROM WorkshopsReservations wr
JOIN DaysReservations dr ON dr.DayReservationID = wr.DayReservationID
WHERE dr.DayReservationID = @DayReservationID AND wr.WorkshopID = @WorkshopID
)
BEGIN
RAISERROR ('Nie można się już zapisać na warsztat', -1, -1)
ROLLBACK TRANSACTION
END
END
GO

```

T_CheckPriceListInsert - Kontroluje poprawność dodania nowego progu cenowego.

```

CREATE TRIGGER T_CheckPriceListInsert
ON PriceList
AFTER INSERT
AS

```

```

BEGIN

    DECLARE @PriceDate DATE
        = (SELECT PriceDate FROM inserted)
    DECLARE @PriceValue money
        = (SELECT PriceValue FROM inserted)
    DECLARE @ConferenceID INT
        = (SELECT ConferenceID FROM inserted)

    IF @PriceDate >= (SELECT StartDate FROM Conferences WHERE ConferenceID = @ConferenceID)
        BEGIN
            RAISERROR ('Data ceny jest późniejsza niż początek konferencji', -1, -1)
            ROLLBACK TRANSACTION
        END

    IF EXISTS (
        SELECT * FROM PriceList
        WHERE PriceDate < @PriceDate AND ConferenceID = @ConferenceID
    )
        BEGIN
            IF @PriceValue <= (
                SELECT TOP 1 PriceValue FROM PriceList
                WHERE PriceDate < @PriceDate AND ConferenceID = @ConferenceID
                ORDER BY PriceDate DESC
            )
                BEGIN
                    RAISERROR ('Cena dla tej daty jest za mała', -1, -1)
                    ROLLBACK TRANSACTION
                END
        END
    END

    IF EXISTS (
        SELECT * FROM PriceList
        WHERE PriceDate > @PriceDate AND ConferenceID = @ConferenceID
    )
        BEGIN
            IF @PriceValue >= (
                SELECT TOP 1 PriceValue FROM PriceList
                WHERE PriceDate > @PriceDate AND ConferenceID = @ConferenceID
                ORDER BY PriceDate
            )
        END

```

```
)  
BEGIN  
    RAISERROR ('Cena dla tej daty jest za duża', -1, -1)  
    ROLLBACK TRANSACTION  
END  
END  
GO
```

T_ControlUpdatingPlacesForWorkshop - Blokuje zmniejszenie liczby miejsc na warsztat jeżeli ilość do tej pory zarezerwowanych miejsc jest większa od nowo podanej liczby miejsc.

```
CREATE TRIGGER T_ControlUpdatingPlacesForWorkshop  
    ON Workshops  
    AFTER UPDATE  
AS  
BEGIN  
  
    DECLARE @WorkshopID      int  
        = ( SELECT WorkshopID FROM inserted)  
  
    DECLARE @NewPlaces      int  
        = ( SELECT Places FROM inserted)  
  
    DECLARE @ReservedPlaces      int  
        = ( SELECT ReservedPlaces FROM F_FreeAndReservedPlacesForWorkshop (@WorkshopID))  
  
    IF @NewPlaces < @ReservedPlaces  
    BEGIN  
        RAISERROR ('Nowa ilosc dostepnych miejsc jest mniejsza od juz zarezerowane  
j.', -1, -1)  
        ROLLBACK TRANSACTION  
    END  
  
END  
GO
```

T_CheckIfWorkshopDayBelongsToConferenceDay - Sprawdza czy warsztat został wpisany na jeden z dni konferencji, do której został przypisany.

```
CREATE TRIGGER T_CheckIfWorkshopDayBelongsToConferenceDay
    ON Workshops
    AFTER INSERT
AS
BEGIN
    DECLARE @ConferenceDay INT
        = ( SELECT datediff(day, StartDate, EndDate)
            FROM Conferences
            WHERE ConferenceID = (
                SELECT ConferenceID
                FROM inserted
            )
        )
    SET @ConferenceDay += 1

    IF @ConferenceDay < (
        SELECT ConferenceDay
        FROM inserted
    )
    BEGIN
        RAISERROR ('Konferencja nie ma tylu dni', -1, -1)
        ROLLBACK TRANSACTION
    END
END
GO
```

T_ControlPlacesForWorkshop - Blokuje rezerwacje lub aktualizacje miejsc na warsztat jeżeli nie ma już tylu wolnych miejsc lub podano więcej miejsc niż zarezerwowano na konferencję.

```
CREATE TRIGGER T_ControlPlacesForWorkshop
    ON WorkshopsReservations
    AFTER UPDATE, INSERT
AS
```

```
BEGIN
```

```
DECLARE @WorkshopID      int  
= ( SELECT WorkshopID FROM inserted)
```

```
DECLARE @IncreaseOfPlaces      int  
= ( SELECT i.NormalReservations - d.NormalReservations  
    FROM inserted AS i  
    INNER JOIN deleted AS d  
    ON i.WorkshopReservationID = d.WorkshopReservationID  
)
```

```
DECLARE @FinalNumbnerOfPlaces      int  
= ( SELECT NormalReservations FROM inserted)
```

```
DECLARE @DayReservationID      int  
= ( SELECT DayReservationID FROM inserted)
```

```
DECLARE @ReservedPlacesForConferenceDay      int  
= ( SELECT NormalReservations + StudentsReservations FROM DaysReservations  
WHERE DayReservationID = @DayReservationID)
```

```
IF @FinalNumbnerOfPlaces > @ReservedPlacesForConferenceDay
```

```
BEGIN
```

```
    RAISERROR ('Nie zarezerwowano tylu miejsc na ten dzien konferencji.', -1, -  
1)
```

```
    ROLLBACK TRANSACTION
```

```
END
```

```
DECLARE @FreePlaces      int
```

```
= ( SELECT FreePlaces FROM F_FreeAndReservedPlacesForWorkshop (@WorkshopID)  
)
```

```
IF @IncreaseOfPlaces > @FreePlaces
```

```
BEGIN
```

```
    RAISERROR ('Nie ma tylu wolnych miejsc na ten warsztat.', -1, -1)
```

```
    ROLLBACK TRANSACTION
```

```
END
```

END

GO

T_CancelAllParticipantWorkshopsReservations2 - Anuluje zapisy na warsztat wszystkich uczestników, którzy są od klienta, który anluwał rezerwację miejsc na ten warsztat.

```
CREATE TRIGGER T_CancelAllParticipantWorkshopsReservations2
    ON WorkshopsReservations
    AFTER UPDATE
AS
BEGIN

    UPDATE ParticipantWorkshops
    SET IsCancelled = 1
    WHERE ParticipantReservationID IN
        (
            SELECT pr.ParticipantReservationID
            FROM ParticipantReservations as pr
            INNER JOIN DaysReservations as dr
                ON pr.DayReservationID = dr.DayReservationID
            INNER JOIN ParticipantWorkshops as pw
                ON pr.ParticipantReservationID = pw.ParticipantReservationID
            WHERE dr.DayReservationID IN
                (
                    SELECT DayReservationID
                    FROM inserted
                    WHERE IsCancelled = 1
                )
        )
    AND IsCancelled = 0 -- bo anulowanie może nastąpić po anulowaniu rezerwacji
    na warsztat, albo po anulowaniu rezerwacji na dzień

END
GO
```

T_ControlFreePlacesReservedByClientForConferenceDay - Blokuje dodanie uczestnika na dzień

konferencji jeżeli zostały wykorzystane miejsca zarezerwowane przez klienta, od którego jest dany uczestnik.

```
CREATE TRIGGER T_ControlFreePlacesReservedByClientForConferenceDay
    ON ParticipantReservations
    AFTER INSERT
AS
BEGIN

    DECLARE @DayReservationID      int
        = ( SELECT DayReservationID FROM inserted)

    DECLARE @IsStudent      bit

    -- sprawdzenie czy dodawany uczestnik to student
    IF ( SELECT StudentCard FROM inserted ) IS NULL
    BEGIN
        SET @IsStudent = 0
    END

    IF ( SELECT StudentCard FROM inserted ) IS NOT NULL
    BEGIN
        SET @IsStudent = 1
    END

    DECLARE @NormalReservations      int
        = ( SELECT NormalReservations FROM DaysReservations WHERE DayReservationID
        = @DayReservationID)

    DECLARE @StudentsReservations      int
        = ( SELECT StudentsReservations FROM DaysReservations WHERE DayReservationI
D = @DayReservationID)

    DECLARE @UsedNormalReservations      int
        = (
            SELECT COUNT(*)
            FROM ParticipantReservations
            WHERE DayReservationID = @DayReservationID
                AND IsCancelled = 0
                AND StudentCard IS NULL
```

```

)
DECLARE @UsedStudentsReservations int
= (
    SELECT COUNT(*)
    FROM ParticipantReservations
    WHERE DayReservationID = @DayReservationID
        AND IsCancelled = 0
        AND StudentCard IS NOT NULL
)
IF @IsStudent = 0 AND @UsedNormalReservations = @NormalReservations
BEGIN
    RAISERROR ('Wszystkie normalne rezerwacje zostaly juz wykorzystane na ten dzien konferencji.', -1, -1)
    ROLLBACK TRANSACTION
END
IF @IsStudent = 1 AND @UsedStudentsReservations = @StudentsReservations
BEGIN
    RAISERROR ('Wszystkie rezerwacje dla studentow zostaly juz wykorzystane na ten dzien konferencji.', -1, -1)
    ROLLBACK TRANSACTION
END
END
GO

```

T_ControlStudentsCardFieldsFilling - Sprawdza czy albo podano obydwa pola identyfikujące studenta, albo nie podano żadnego.

```

CREATE TRIGGER T_ControlStudentsCardFieldsFilling
    ON ParticipantReservations
    AFTER INSERT, UPDATE
AS
BEGIN

```

```

DECLARE @StudentCard      int
= ( SELECT StudentCard FROM inserted )

DECLARE @StudentCardDate    date
= ( SELECT StudentCardDate FROM inserted )

IF (@StudentCard IS NULL AND @StudentCardDate IS NOT NULL)
OR (@StudentCard IS NOT NULL AND @StudentCardDate IS NULL)
BEGIN
    RAISERROR ('Proszę wypełnić wszystkie pola przeznaczone dla studenta.', -
1, -1)
    ROLLBACK TRANSACTION
END
END
GO

```

T_ControlUpdatingPlacesForWorkshop - Blokuje zmniejszenie liczby miejsc na warsztat jeżeli ilość do tej pory zarezerwowanych miejsc jest większa od nowo podanej liczby miejsc.

```

CREATE TRIGGER T_ControlUpdatingPlacesForWorkshop
ON Workshops
AFTER UPDATE
AS
BEGIN

DECLARE @WorkshopID      int
= ( SELECT WorkshopID FROM inserted)

DECLARE @NewPlaces      int
= ( SELECT Places FROM inserted)

DECLARE @ReservedPlaces      int
= ( SELECT ReservedPlaces FROM F_FreeAndReservedPlacesForWorkshop (@WorkshopID))

IF @NewPlaces < @ReservedPlaces
BEGIN
    RAISERROR ('Nowa ilość dostępnych miejsc jest mniejsza od już zarezerwowane

```

```
j.', -1, -1)  
      ROLLBACK TRANSACTION  
    END  
  
END  
GO
```

T_DeleteFineAssesdAfterCancelingConferenceReservation - Zeruje należną opłatę po anulowaniu przez klienta rezerwacji na konferencję.

```
CREATE TRIGGER T_DeleteFineAssesdAfterCancelingConferenceReservation  
  ON ClientReservations  
  AFTER UPDATE  
  AS  
BEGIN  
  DECLARE @ClientReservationID      int  
        = ( SELECT ClientReservationID FROM inserted WHERE IsCancelled = 1)  
  
  IF @ClientReservationID IS NOT NULL  
  BEGIN  
    UPDATE Payments  
    SET FineAssessed = 0  
    WHERE PaymentID = @ClientReservationID  
  END  
  
END  
GO
```

T_CountFineAfterWorkhopReservationOrUpdate - Wylicza należną opłatę za rezerwację na konferencję i warsztaty po zarezerwowaniu lub anulowaniu miejsc na dany warsztat.

```
CREATE TRIGGER T_CountFineAfterWorkhopReservationOrUpdate  
  ON WorkshopsReservations  
  AFTER INSERT, UPDATE  
  AS  
BEGIN
```

```

DECLARE @DayReservationID      int
= ( SELECT DayReservationID FROM inserted)

DECLARE @ClientReservationID   int
= ( SELECT ClientReservationID FROM DaysReservations WHERE DayReservationID
= @DayReservationID)

EXEC P_CountFine @ClientReservationID = @ClientReservationID;

END
GO

```

T_CountFineAfterConferenceDayReservationOrUpdate - Wylicza należną opłatę za rezerwację na konferencję i warsztaty po zarezerwowaniu lub anulowaniu miejsc na dany dzień konferencji.

```

CREATE TRIGGER T_CountFineAfterConferenceDayReservationOrUpdate
    ON DaysReservations
    AFTER INSERT, UPDATE
AS
BEGIN

    DECLARE @ClientReservationID      int
        = ( SELECT ClientReservationID FROM inserted)

    EXEC P_CountFine @ClientReservationID = @ClientReservationID;

END
GO

```

T_CreatePaymentField - tworzy wpis w tabeli Payments po rejestracji klienta na daną konferencję

```

CREATE TRIGGER T_CreatePaymentField
    ON dbo.ClientReservations
    AFTER INSERT
AS

```

```
BEGIN
```

```
DECLARE @PaymentID      INT  
= (  
    SELECT ClientReservationID  
    FROM Inserted  
)
```

```
DECLARE @ReservationDate DATE  
= (  
    SELECT ReservationDate  
    FROM Inserted  
)
```

```
DECLARE @DueDate      DATE  
= DATEADD(DAY, 7, @ReservationDate)
```

```
INSERT INTO dbo.Payments  
(  
    PaymentID,  
    FineAssessed,  
    FinePaid,  
    DueDate  
)  
VALUES  
(  
    @PaymentID,  
    0,      -- FineAssessed - money  
    0,      -- FinePaid - money  
    @DueDate -- DueDate - date  
)
```

```
END
```

```
GO
```

Procedure

P_AddClient - dodaje nowego klienta

```
CREATE PROCEDURE P_AddClient
    @ClientName VARCHAR(50),
    @ClientSurname VARCHAR(50),
    @IsPrivate BIT,
    @PhoneNumber INT,
    @Email VARCHAR(50),
    @Address VARCHAR(50),
    @City VARCHAR(50),
    @PostalCode INT,
    @Country VARCHAR(30)

AS
BEGIN
    IF EXISTS (
        SELECT * FROM Clients
        WHERE @Email = Email
    )
    BEGIN
        RAISERROR ('Taki klient już istnieje w bazie', -1, -1)
        RETURN
    END

    INSERT INTO Clients
        VALUES (
            @ClientName,
            @ClientSurname,
            @IsPrivate,
            @PhoneNumber,
            @Email,
            @Address,
            @City,
            @PostalCode,
            @Country
        )
END
GO
```

P_AddConference - dodaje nową konferencję

```
CREATE PROCEDURE P_AddConference
    @ConferenceName varchar(50),
    @StartDate date,
    @EndDate date,
    @Places int,
    @Discount float(10)

AS
BEGIN
    INSERT INTO Conferences
        VALUES
    (
        @ConferenceName,
        @StartDate,
        @EndDate,
        @Places,
        @Discount
    )
END
GO

--dodawanie uczestników
CREATE PROCEDURE P_AddParticipant
    @Name VARCHAR(50),
    @Surname VARCHAR(50),
    @PhoneNumber INT,
    @Email VARCHAR(50),
    @City VARCHAR(50),
    @Country VARCHAR(50)

AS
BEGIN
    IF EXISTS (
        SELECT * FROM Participants
        WHERE Email = @Email
    )
    BEGIN
        RAISERROR ('Taki uczestnik już istnieje', -1, -1)
        RETURN
    END

```

```
INSERT INTO Participants
    VALUES (
        @Name,
        @Surname,
        @PhoneNumber,
        @Email,
        @City,
        @Country
    )
END
GO
```

P_AddParticipantForConferenceDay - dodaje uczestnika na dany dzień konferencji

```
CREATE PROCEDURE P_AddParticipantForConferenceDay
(
    @ParticipantID int,
    @DayReservationID int,
    @StudentCard int,
    @StudentCardDate date
)
AS
BEGIN
    -- czy uczestnik juz nie zostal zarejestrowany
    IF EXISTS
    (
        SELECT *
        FROM ParticipantReservations
        WHERE ParticipantID      = @ParticipantID
            AND DayReservationID = @DayReservationID
            AND IsCancelled      = 0
    )
    BEGIN
        RAISERROR ('Podany uczestnik jest ju? zarejestrowany na ten dzie? konferencji.', -1, -1)
        RETURN
    END
END
```

```

-- czy uczestnik istnieje
IF NOT EXISTS
(
    SELECT *
    FROM Participants
    WHERE ParticipantID = @ParticipantID
)
BEGIN
    RAISERROR ('Nie ma uczestnika o podanym ID.', -1, -1)
    RETURN
END

-- czy istnieje rezerwacja dnia o podanym ID
IF NOT EXISTS
(
    SELECT *
    FROM DaysReservations
    WHERE DayReservationID = @DayReservationID
        AND IsCancelled = 0
)
BEGIN
    RAISERROR ('Nie ma rezerwacji dnia o podanym ID.', -1 , -1)
    RETURN
END

INSERT INTO ParticipantReservations
(
    ParticipantID,
    DayReservationID,
    StudentCard,
    StudentCardDate
)
VALUES
(
    @ParticipantID,
    @DayReservationID,
    @StudentCard,
    @StudentCardDate
)

```

END

GO

P_AddParticipantForWorkshop - dodaje uczestnika na dany warsztat

```
CREATE PROCEDURE P_AddParticipantForWorkshop
    @ParticipantReservationID int,
    @WorkshopID int
AS
BEGIN
    IF EXISTS
        (
            SELECT *
            FROM ParticipantWorkshops
            WHERE ParticipantReservationID = @ParticipantReservationID
                AND WorkshopID = @WorkshopID
                AND IsCancelled = 0
        )
        BEGIN
            RAISERROR ('Podany uczestnik rezerwowany jest już zarejestrowany na ten warsztat.', -1, -1)
            RETURN
        END
    INSERT INTO ParticipantWorkshops
        (
            ParticipantReservationID,
            WorkshopID
        )
    VALUES
        (
            @ParticipantReservationID,
            @WorkshopID
        )
END
GO
```

```

-- dodawanie progu cenowego
CREATE PROCEDURE P_AddPriceToConferencePriceList
    @ConferenceID int,
    @PriceValue money,
    @PriceDate date
AS
BEGIN
    IF NOT EXISTS
        (
            SELECT *
            FROM Conferences
            WHERE ConferenceID = @ConferenceID
        )
    BEGIN
        RAISERROR ('Nie ma konferencji o takim ID.', -1, -1)
        RETURN
    END

    INSERT INTO PriceList
        VALUES
        (
            @ConferenceID,
            @PriceValue,
            @PriceDate
        )
END
GO

```

P_AddReservationForConference - dodaje rezerwacje danego klienta na daną rezerwację

```

CREATE PROCEDURE P_AddReservationForConference
    @ConferenceID int,
    @ClientID int
AS
BEGIN
    IF EXISTS
        (

```

```

SELECT *
FROM ClientReservations
WHERE ConferenceID = @ConferenceID
AND ClientID = @ClientID
AND IsCancelled = 0
)
BEGIN
RAISERROR ('Klient o podanym ID juz rejestrowas sie na konferencje o poda-
nym ID', -1, -1)
RETURN
END
INSERT INTO ClientReservations (ConferenceID, ClientID)
VALUES
(
    @ConferenceID,
    @ClientID
)
END
GO

```

P_AddReservationForConferenceDay - dodaje rezerwacje danego klienta na dany dzień konferencji

```

CREATE PROCEDURE P_AddReservationForConferenceDay
    @ClientReservationID int,
    @ConferenceDay int,
    @NormalReservations int,
    @StudentReservations int
AS
BEGIN
IF EXISTS
(
    SELECT *
    FROM DaysReservations
    WHERE ClientReservationID = @ClientReservationID
        AND ConferenceDay      = @ConferenceDay
        AND IsCancelled         = 0
)

```

```

BEGIN
    RAISERROR ('Ten klient dokonywał już rejestracji na ten dzień konferencji',
    -1, -1)
    RETURN
END

INSERT INTO DaysReservations
(
    ClientReservationID,
    ConferenceDay,
    NormalReservations,
    StudentsReservations
)
VALUES
(
    @ClientReservationID,
    @ConferenceDay,
    @NormalReservations,
    @StudentReservations
)

END
GO

```

P_AddReservationForWorkshop - dodaje rezerwacje danego klienta na dany warsztat

```

CREATE PROCEDURE P_AddReservationForWorkshop
    @DayReservationID int,
    @WorkshopID int,
    @NormalReservations int

AS
BEGIN
    IF EXISTS
        (
            SELECT *
            FROM WorkshopsReservations
            WHERE DayReservationID = @DayReservationID
                AND WorkshopID      = @WorkshopID
        )

```

```

        AND IsCancelled      = 0
    )
BEGIN
    RAISERROR ('Ten klient rezerwował juz miejsca na ten warsztat', -1, -1)
    RETURN
END

INSERT INTO WorkshopsReservations
(
    DayReservationID,
    WorkshopID,
    NormalReservations
)
VALUES
(
    @DayReservationID,
    @WorkshopID,
    @NormalReservations
)
END
GO

```

P_AddWorkshop - dodaje nowy warsztat

```
CREATE PROCEDURE P_AddWorkshop
```

```

    @ConferenceID int,
    @ConferenceDay int,
    @WorkshopName varchar(50),
    @Places int,
    @WorkshopFee money,
    @WorkshopStart time,
    @WorkshopEnd time

```

```
AS
```

```
BEGIN
```

```
    INSERT INTO Workshops
```

```
        VALUES
```

```
(
```

```
    @ConferenceID,
```

```
        @ConferenceDay,  
        @WorkshopName,  
        @Places,  
        @WorkshopFee,  
        @WorkshopStart,  
        @WorkshopEnd  
    )  
END  
GO
```

P_CancelConferenceReservation - anuluje rezerwacje na konferencję

```
CREATE PROCEDURE P_CancelConferenceReservation  
    @ClientReservationID      int  
AS  
BEGIN  
  
    -- czy istnieje rezerwacja o podanym id  
    IF NOT EXISTS  
    (  
        SELECT *  
        FROM ClientReservations  
        WHERE ClientReservationID = @ClientReservationID  
    )  
    BEGIN  
        RAISERROR ('Nie ma rezerwacji na konferencje o podanym ID.', -1, -1)  
        RETURN  
    END  
  
    -- czy rezerwacja zostala juz anulowana  
    IF  
    (  
        SELECT IsCancelled  
        FROM ClientReservations  
        WHERE ClientReservationID = @ClientReservationID  
    ) = 1  
    BEGIN  
        RAISERROR ('Ta rezerwacja została już anulowana.', -1, -1)
```

```

    RETURN
END

UPDATE ClientReservations
    SET IsCancelled = 1
    WHERE ClientReservationID = @ClientReservationID

END
GO

```

P_CabcekDayReservation - anuluje daną rezerwację na dzień konferencji

```

CREATE PROCEDURE P_CancelDayReservation
    @DayReservationID int
AS
BEGIN
    IF NOT EXISTS
        (
            SELECT * FROM DaysReservations WHERE DayReservationID = @DayReservationID
        )
    BEGIN
        RAISERROR ('Nie istnieje taki dzień', -1, -1)
        RETURN
    END

    IF (
        SELECT IsCancelled FROM DaysReservations WHERE DayReservationID = @DayReservationID
        ) = 1
    BEGIN
        RAISERROR ('Rezerwacja jest już anulowana', -1, -1)
        RETURN
    END

    UPDATE DaysReservations
        SET IsCancelled = 1
        WHERE DayReservationID = @DayReservationID

```

END

GO

P_CancelParticipantReservation - anuluje rezerwacje uczestnika na dany dzień konferencji

```
CREATE PROCEDURE P_CancelParticipantReservation
    @ParticipantReservationID INT
AS
BEGIN
    IF NOT EXISTS
        (
            SELECT * FROM ParticipantReservations WHERE ParticipantReservationID =
@ParticipantReservationID
        )
    BEGIN
        RAISERROR ('Nie istnieje taka rezerwacja', -1, -1)
        RETURN
    END

    IF (
        SELECT IsCancelled FROM ParticipantReservations WHERE ParticipantReservatio
nID = @ParticipantReservationID
        ) = 1
    BEGIN
        RAISERROR ('Rezerwacja jest już anulowana', -1, -1)
        RETURN
    END

    UPDATE ParticipantReservations
    SET IsCancelled = 1
    WHERE ParticipantReservationID = @ParticipantReservationID
END
GO
```

P_CancelParticipantWorkshopReservation - anuluje rezerwację uczestnika na dany warsztat

```

CREATE PROCEDURE P_CancelParticipantWorkshopReservation
    @WorkshopReservationID INT
AS
BEGIN
    IF NOT EXISTS (
        SELECT * FROM ParticipantWorkshops WHERE WorkshopReservationID = @WorkshopReservationID
    )
    BEGIN
        RAISERROR ('Nie istnieje taka rezerwacja', -1, -1)
        RETURN
    END

    IF (
        SELECT IsCancelled FROM ParticipantWorkshops WHERE WorkshopReservationID =
        @WorkshopReservationID
        ) = 1
    BEGIN
        RAISERROR ('Rezerwacja jest już anulowana', -1, -1)
        RETURN
    END

    UPDATE ParticipantWorkshops
    SET IsCancelled = 1
    WHERE WorkshopReservationID = @WorkshopReservationID
END
GO

```

P_CancelUnpaidReservation - anuluje nieopłacone w terminie rezerwacje

```

CREATE PROCEDURE P_CancelUnpaidReservation
AS
BEGIN
    DECLARE @ClientReservationID INT;

    WHILE EXISTS (
        SELECT * FROM Payments p

```

```

        JOIN ClientReservations cr ON cr.ClientReservationID = p.PaymentID
        WHERE p.FinePaid < p.FineAssessed AND p.DueDate < convert(date, getdate())
AND cr.IsCancelled = 0
)
BEGIN
    SET @ClientReservationID = (
        SELECT TOP 1 cr.ClientReservationID FROM Payments p
        JOIN ClientReservations cr ON cr.ClientReservationID = p.PaymentID
        WHERE p.FinePaid < p.FineAssessed AND p.DueDate < convert(date, getdate()
()) AND cr.IsCancelled = 0
);
EXEC P_CancelConferenceReservation @ClientReservationID;
END
END
GO

```

P_CancelWorkshopReservation - anuluje rezerwacje klienta na dany warsztat

```

CREATE PROCEDURE P_CancelWorkshopReservation
@WorkshopReservationID INT
AS
BEGIN
IF NOT EXISTS (
    SELECT * FROM WorkshopsReservations
    WHERE WorkshopReservationID = @WorkshopReservationID
)
BEGIN
    RAISERROR ('Taka rezerwacja nie istnieje', -1, -1)
    RETURN
END

IF (
    SELECT IsCancelled FROM WorkshopsReservations
    WHERE WorkshopReservationID = @WorkshopReservationID
) = 1
BEGIN
    RAISERROR ('Ta rezerwacja jest już anulowana', -1, -1)

```

```

    RETURN
END

UPDATE WorkshopsReservations
    SET IsCancelled = 1
    WHERE WorkshopReservationID = @WorkshopReservationID

END
GO

```

P_ChangeConferenceDetails - zmienia dane konferencji, w tym ilość dostępnych miejsc

```

CREATE PROCEDURE P_ChangeConferenceDetails
    @ConferenceID int,
    @StartDate date,
    @EndDate date,
    @Places int,
    @Discount float(10)

AS
BEGIN
    IF NOT EXISTS
        (
            SELECT *
            FROM Conferences
            WHERE ConferenceID = @ConferenceID
        )
        BEGIN
            RAISERROR ('Nie ma konferencji o takim ID.', -1, -1)
            RETURN
        END
-- aktualizowanie poczatku konferencji
    IF @StartDate IS NOT NULL
        BEGIN
            UPDATE Conferences
                SET StartDate      = @StartDate
                WHERE ConferenceID = @ConferenceID
        END

```

```

-- aktualizowanie konca konferencji
IF @EndDate IS NOT NULL
BEGIN
    UPDATE Conferences
        SET EndDate      = @EndDate
    WHERE ConferenceID = @ConferenceID
END

-- aktualizowanie zniżki
IF @Discount IS NOT NULL
BEGIN
    UPDATE Conferences
        SET Discount      = @Discount
    WHERE ConferenceID = @ConferenceID
END

-- aktualizowanie miejsc
IF @Places IS NOT NULL
BEGIN
    UPDATE Conferences
        SET Places        = @Places
    WHERE ConferenceID = @ConferenceID
END

END
GO

```

P_ChangeDayReservationPlaces - zmienia ilość zarezerwowanych przez klienta miejsc na dany dzień konferencji

```

CREATE PROCEDURE P_ChangeDayReservationPlaces
    @DayReservationID INT,
    @NumberOfPlaces INT,
    @IsStudent BIT
AS
BEGIN
    IF @IsStudent = 0

```

```

BEGIN
    UPDATE DaysReservations
        SET NormalReservations = @NumberOfPlaces
        WHERE DayReservationID = @DayReservationID
    RETURN
END

UPDATE DaysReservations
    SET StudentsReservations = @NumberOfPlaces
    WHERE DayReservationID = @DayReservationID
END
GO

```

P_ChangeWorkshopDetails - zmienia dane warsztatu w tym ilość dostępnych miejsc

```

CREATE PROCEDURE P_ChangeWorkshopDetails
    @WorkshopID int,
    @ConferenceDay int,
    @Places int,
    @WorkshopStart time,
    @WorkshopEnd time

AS
BEGIN
    IF NOT EXISTS
        (
            SELECT *
            FROM Workshops
            WHERE WorkshopID = @WorkshopID
        )
    BEGIN
        RAISERROR ('Nie ma warsztatu o takim ID.', -1, -1)
        RETURN
    END

    -- aktualizowanie dnia
    IF @ConferenceDay IS NOT NULL
    BEGIN
        UPDATE Workshops

```

```

SET ConferenceDay = @ConferenceDay
WHERE WorkshopID = @WorkshopID
END

-- aktualizownaie ilości miejsc
IF @Places IS NOT NULL
BEGIN
    UPDATE Workshops
        SET Places          = @Places
        WHERE WorkshopID = @WorkshopID
END

-- aktualizowanie czasu rozpoczęcia
IF @WorkshopStart IS NOT NULL
BEGIN
    UPDATE Workshops
        SET WorkshopStart = @WorkshopStart
        WHERE WorkshopID = @WorkshopID
END

-- aktualizowanie czasu zakończenia
IF @WorkshopEnd IS NOT NULL
BEGIN
    UPDATE Workshops
        SET WorkshopEnd   = @WorkshopEnd
        WHERE WorkshopID = @WorkshopID
END

END
GO

```

P_ChangeWorkshopReservationPlaces - zmienia ilość zarezerwowanych przez klienta miejsc na dany warsztat

```

CREATE PROCEDURE P_ChangeWorkshopReservationPlaces
    @WorkshopReservationID INT,
    @NumberOfPlaces INT
AS

```

```
BEGIN  
    UPDATE WorkshopReservations  
    SET NormalReservations = @NumberOfPlaces  
    WHERE WorkshopReservationID = @WorkshopReservationID  
END  
GO
```

P_CheckCurrentPayment - sprawdza status opłaty klienta

```
CREATE PROCEDURE P_CheckCurrentPayment  
    @ClientID INT,  
    @ConferenceID INT  
AS  
BEGIN  
    DECLARE @PaymentID INT, @FineAssessed money, @FinePaid money;  
    SET @PaymentID = (  
        SELECT ClientReservationID  
        FROM ClientReservations  
        WHERE @ConferenceID = ConferenceID AND @ClientID = ClientID  
    )  
  
    SET @FineAssessed = (  
        SELECT FineAssessed  
        FROM Payments  
        WHERE @PaymentID = PaymentID  
    )  
    SET @FinePaid = (  
        SELECT FinePaid  
        FROM Payments  
        WHERE @PaymentID = PaymentID  
    )  
  
    IF @FineAssessed > @FinePaid  
    BEGIN  
        PRINT 'Klient jeszcze nie zapłacił'  
        RETURN  
    END
```

```

IF @FineAssessed = @FinePaid
BEGIN
    PRINT 'Klient zapłacił'
    RETURN
END

PRINT 'Klient nadpłacił'
END
GO

```

P_CountFine - wylicza opłatę należną za rezerwacje miejsc na konferencję i warsztaty dla konkretnej rezerwacji

```

CREATE PROCEDURE P_CountFine
    @ClientReservationID INT
AS
BEGIN
    DECLARE @CurrentPrice money, @Sum money, @Discount float(10);
    DECLARE @ConferenceID INT
        = ( SELECT ConferenceID
            FROM ClientReservations
            WHERE ClientReservationID = @ClientReservationID )
    SET @CurrentPrice = dbo.F_GetCurrentPrice(@ConferenceID, (
        SELECT ReservationDate
        FROM ClientReservations
        WHERE @ClientReservationID = ClientReservationID
    ));
    SET @Discount = (
        SELECT Discount
        FROM Conferences
        WHERE ConferenceID = @ConferenceID
    );
    SET @Sum = (
        SELECT ((SUM(NormalReservations) + (SUM(StudentsReservations) * (1 - @Discount))) * @CurrentPrice) as NumberOfPlaces
        FROM DaysReservations
        WHERE @ClientReservationID = ClientReservationID AND IsCancelled = 0
    );

```

```

        GROUP BY ClientReservationID
) + (
    SELECT SUM(wr.NormalReservations * w.WorkshopFee)
    FROM WorkshopsReservations wr
    JOIN DaysReservations dr ON dr.DayReservationID = wr.DayReservationID
    JOIN Workshops w ON w.WorkshopID = wr.WorkshopID
    WHERE dr.ClientReservationID = @ClientReservationID AND wr.IsCancelled = 0
    GROUP BY dr.ClientReservationID
);

UPDATE Payments
    SET FineAssessed = @Sum
    WHERE PaymentID = @ClientReservationID
END
GO

```

P_DeletePriceFromConferencePriceList - usuwa dany próg cenowy

```

CREATE PROCEDURE P_DeletePriceFromConferencePriceList
    @PriceID int
AS
BEGIN
    IF NOT EXISTS
    (
        SELECT *
        FROM PriceList
        WHERE PriceID = @PriceID
    )
    BEGIN
        RAISERROR ('Nie ma progu cenowego o podanym ID', -1, -1)
        RETURN
    END
    DELETE PriceList
    WHERE PriceID = @PriceID
END
GO

```

Funkcje

F_AllPaymentsByClientID - zwraca tabelę z całą historią płatności klienta (w tym te nieuregulowane)

```
CREATE FUNCTION F_AllPaymentsByClientID
(
    @ClientID int
)
RETURNS TABLE
AS
RETURN
    SELECT c.ClientID, c.ClientName, c.ClientSurname, conf.ConferenceID, conf.C
onferenceName, p.FineAssessed, p.FinePaid
        FROM Clients AS c
        INNER JOIN ClientReservations AS cr
            ON c.ClientID = cr.ClientID
        INNER JOIN Payments AS p
            ON cr.ClientReservationID = p.PaymentID
        INNER JOIN Conferences AS conf
            ON cr.ConferenceID = conf.ConferenceID
    WHERE c.ClientID = @ClientID
        AND cr.IsCancelled = 0
GO
```

F_ClientReservationsHistory - zwraca tabelę z całą historią rezerwacji klienta (w tym obecne)

```
CREATE FUNCTION F_ClientReservationsHistory
(
    @ClientID int
)
RETURNS TABLE
AS
RETURN
    SELECT c.ClientID, c.ClientName, c.ClientSurname, cr.ConferenceID, cr.IsCan
celled, conf.ConferenceName
        FROM Clients AS c
        INNER JOIN ClientReservations AS cr
```

```
    ON c.ClientID = cr.ClientID
  INNER JOIN Conferences AS conf
    ON cr.ConferenceID = conf.ConferenceID
  WHERE c.ClientID = @ClientID
```

```
GO
```

F_ClientsWithUnusedPlaces - zwraca tabelę przedstawiającą listę klientów wraz z liczbą niewykorzystanych jeszcze przez uczestników zgłoszonych przez danego klienta miejsc

```
CREATE FUNCTION F_ClientsWithUnusedPlaces
(
  @ConferenceID int
)
RETURNS TABLE
AS
RETURN
  SELECT dr.ConferenceDay, c.ClientName, (dr.NormalReservations + dr.StudentsRese
rvations - (
    SELECT COUNT(*)
    FROM ParticipantReservations pr
    WHERE pr.DayReservationID = dr.DayReservationID
  )) as FreePlaces
  FROM DaysReservations dr
  JOIN ClientReservations cr
    ON cr.ClientReservationID = dr.ClientReservationID
  JOIN Clients c
    ON c.ClientID = cr.ClientID
  WHERE cr.ConferenceID = @ConferenceID AND (dr.NormalReservations + dr.StudentsR
eservations - (
    SELECT COUNT(*)
    FROM ParticipantReservations pr
    WHERE pr.DayReservationID = dr.DayReservationID
  )) != 0
GO
```

F_ConferenceParticipants - zwraca listę wszystkich uczestników biorących udział w danej konferencji

```

CREATE FUNCTION F_ConferenceParticipants
(
    @ConferenceID int
)
RETURNS TABLE
AS
RETURN
    SELECT DISTINCT sub.ParticipantID, p.Name, p.Surname
    FROM (
        SELECT pr.ParticipantID
        FROM Conferences c
            JOIN ClientReservations cr
            ON c.ConferenceID = cr.ConferenceID
            JOIN DaysReservations dr
            ON dr.ClientReservationID = cr.ClientReservationID
            JOIN ParticipantReservations pr
            ON pr.DayReservationID = dr.DayReservationID
        WHERE c.ConferenceID = @ConferenceID AND pr.IsCancelled = 0
    ) as sub
    JOIN Participants p
    ON p.ParticipantID = sub.ParticipantID
GO

```

F_CreatePeopleIdentifiers - zwraca tabelę przedstawiającą listę identyfikatorów uczestników biorących udział w danej konferencji

```

CREATE FUNCTION F_CreatePeopleIdentifiers
(
    @ConferenceID int
)
RETURNS TABLE
AS
RETURN
    SELECT DISTINCT sub.ParticipantID, p.Name, p.Surname, IIF(c.IsPrivate = 0, '', c.ClientName) as ClientName
    FROM (
        SELECT pr.ParticipantID, cr.ClientID
        FROM Conferences c

```

```

        JOIN ClientReservations cr
        ON c.ConferenceID = cr.ConferenceID
        JOIN DaysReservations dr
        ON dr.ClientReservationID = cr.ClientReservationID
        JOIN ParticipantReservations pr
        ON pr.DayReservationID = dr.DayReservationID
        WHERE c.ConferenceID = @ConferenceID AND pr.IsCancelled = 0
    ) AS sub
    JOIN Participants p
    ON p.ParticipantID = sub.ParticipantID
    JOIN Clients c
    ON c.ClientID = sub.ClientID
GO

```

F_FreeAndReservedPlacesForConference - zwraca tabelę przedstawiającą listę dni konferencji wraz z liczbą wszystkich, wolnych i zarezerwowanych miejsc

```

CREATE FUNCTION F_FreeAndReservedPlacesForConference
(
    @ConferenceID  int
)
RETURNS TABLE
AS
RETURN
    SELECT c.ConferenceName,
           dr.ConferenceDay,
           c.Places,
           c.Places - SUM(dr.NormalReservations + dr.StudentsReservations) AS FreePlaces,
           SUM(dr.NormalReservations + dr.StudentsReservations) AS ReservedPlaces
    FROM Conferences AS c
    INNER JOIN ClientReservations AS cr
        ON c.ConferenceID = cr.ConferenceID
    INNER JOIN DaysReservations AS dr
        ON cr.ClientReservationID = dr.ClientReservationID
    WHERE c.ConferenceID = @ConferenceID

```

```
        AND dr.IsCancelled = 0  
    GROUP BY dr.ConferenceDay, c.Places, c.ConferenceName
```

GO

F_FreeAndReservedPlacesForWorkshop - zwraca tabelę przedstawiającą listę warsztatów wraz z liczbą wszystkich, wolnych i zaerzerwowanych miejsc

```
CREATE FUNCTION F_FreeAndReservedPlacesForWorkshop  
(  
    @WorkshopID int  
)  
RETURNS TABLE  
AS  
RETURN  
    SELECT w.WorkshopID,  
          w.WorkshopName,  
          w.Places,  
          SUM(wr.NormalReservations) AS ReservedPlaces,  
          w.Places - SUM(wr.NormalReservations) AS FreePlaces  
    FROM Workshops AS w  
    INNER JOIN WorkshopsReservations AS wr  
        ON w.WorkshopID = wr.WorkshopID  
    WHERE wr.IsCancelled = 0  
    AND w.WorkshopID = @WorkshopID  
    GROUP BY w.WorkshopID, w.Places, w.WorkshopName
```

GO

F_NonregulatedPaymentsByClientID - zwraca tabelę przedstawiającą listę nieuregulowanych opłat danego klienta

```
CREATE FUNCTION F_NonregulatedPaymentsByClientID  
(  
    @ClientID int  
)  
RETURNS TABLE  
AS
```

```

RETURN
SELECT c.ClientID, c.ClientName, c.ClientSurname, conf.ConferenceID, conf.C
onferenceName, p.FineAssessed, p.FinePaid
FROM Clients AS c
INNER JOIN ClientReservations AS cr
    ON c.ClientID = cr.ClientID
INNER JOIN Payments AS p
    ON cr.ClientReservationID = p.PaymentID
INNER JOIN Conferences AS conf
    ON cr.ConferenceID = conf.ConferenceID
WHERE cr.IsCancelled = 0
    AND p.FinePaid < p.FineAssessed
    AND c.ClientID = @ClientID

```

GO

F_ParticipantsListForConferenceDay - zwraca tabelę przedstawiającą listę uczestników zapisanych na dany dzień konferencji

```

CREATE FUNCTION F_ParticipantsListForConferenceDay
(
    @ConferenceID int,
    @ConferenceDay int
)
RETURNS TABLE
AS
RETURN
SELECT c.ConferenceName, dr.ConferenceDay, p.*
FROM Participants AS p
INNER JOIN ParticipantReservations as pr
    ON p.ParticipantID = pr.ParticipantID
INNER JOIN DaysReservations AS dr
    ON pr.DayReservationID = dr.DayReservationID
INNER JOIN ClientReservations AS cr
    ON cr.ClientReservationID = dr.ClientReservationID
INNER JOIN Conferences AS c
    ON cr.ConferenceID = c.ConferenceID
WHERE c.ConferenceID = @ConferenceID
    AND dr.ConferenceDay = @ConferenceDay

```

```
        AND pr.IsCancelled = 0
```

```
GO
```

F_ParticipantsListForWorkshop - zwraca listę uczestników zapisanych na dany warsztat

```
CREATE FUNCTION F_ParticipantsListForWorkshop
(
    @WorkshopID int
)
RETURNS TABLE
AS
RETURN
    SELECT w.WorkshopID, w.WorkshopName, p.*
    FROM Participants AS p
    INNER JOIN ParticipantReservations AS pr
        ON p.ParticipantID = pr.ParticipantID
    INNER JOIN ParticipantWorkshops AS pw
        ON pr.ParticipantReservationID = pw.ParticipantReservationID
    INNER JOIN Workshops AS w
        ON w.WorkshopID = pw.WorkshopID
    WHERE w.WorkshopID = @WorkshopID
        AND pw.IsCancelled = 0
```

```
GO
```

F_RegulatedPaymentsByClientID - zwraca tabelę przedstawiającą historię uregulowanych opłat danego klienta

```
CREATE FUNCTION F_RegulatedPaymentsByClientID
(
    @ClientID int
)
RETURNS TABLE
AS
RETURN
    SELECT c.ClientID, c.ClientName, c.ClientSurname, conf.ConferenceID, conf.ConferenceName, p.FineAssessed, p.FinePaid
```

```
FROM Clients AS c
INNER JOIN ClientReservations AS cr
    ON c.ClientID = cr.ClientID
INNER JOIN Payments AS p
    ON cr.ClientReservationID = p.PaymentID
INNER JOIN Conferences AS conf
    ON cr.ConferenceID = conf.ConferenceID
WHERE c.ClientID = @ClientID
    AND cr.IsCancelled = 0
    AND p.FineAssessed <= FinePaid
```

GO

F_ShowPrices - zwraca tabelę przedstawiającą cennik dotyczący rezerwacji miejsc na konferencję w zależności od daty wykonania rezerwacji

```
CREATE FUNCTION F_ShowPrices
(
    @ConferenceID int
)
RETURNS TABLE
AS
RETURN
    SELECT p.PriceValue, p.PriceDate
    FROM PriceList p
    WHERE p.ConferenceID = @ConferenceID
GO
```

F_ShowWorkshops - zwraca tabelę przedstawiającą listę warsztatów odbywających się podczas danej konferencji

```
CREATE FUNCTION F_ShowWorkshops
(
    @ConferenceID int
)
RETURNS TABLE
AS
```

```
RETURN
```

```
SELECT WorkshopID, WorkshopName  
FROM Workshops  
WHERE ConferenceID = @ConferenceID
```

```
GO
```

F_GetCurrentPrice - zwraca wysokość opłaty za rezerwację na konferencję gdyby miała nastąpić w tym momencie

```
CREATE FUNCTION F_GetCurrentPrice  
(  
    @ConferenceID int,  
    @CurrentDate date  
)  
RETURNS money  
AS  
BEGIN  
    DECLARE @Price money;  
    SELECT TOP 1 @Price = PriceValue  
    FROM PriceList  
    WHERE @ConferenceID = ConferenceID AND @CurrentDate <= PriceDate  
    ORDER BY PriceDate;  
    RETURN @Price  
END  
GO
```

Widoki

V_MostFrequentClients - przedstawia listę 10 klientów najczęściej dokonujących rezerwacji na konferencje

```
CREATE VIEW V_MostFrequentClients  
AS  
SELECT TOP 10 c.ClientName, c.ClientSurname, COUNT (*) AS Frequency  
FROM ClientReservations as cr
```

```
INNER JOIN Clients AS c
    ON cr.ClientID = c.ClientID
WHERE cr.IsCancelled = 0
GROUP BY cr.ClientID, c.ClientName, c.ClientSurname
ORDER BY Frequency DESC
GO
```

V_MostProfitableClients - przedstawia listę 10 klientów, którzy najwięcej zapłacili za rezerwacje miejsc na konferencje i warsztaty od początku działalności firmy

```
CREATE VIEW V_MostProfitableClients
AS
SELECT TOP 10 c.ClientName, c.ClientSurname, SUM(p.FineAssessed) as TotalProfit
FROM Payments as p
INNER JOIN ClientReservations as cr
    ON p.PaymentID = cr.ClientReservationID
INNER JOIN Clients as c
    ON cr.ClientID = c.ClientID
WHERE cr.IsCancelled = 0
GROUP BY cr.ClientID, c.ClientName, c.ClientSurname
ORDER BY TotalProfit DESC
GO
```

V_UnpayedCancelledReservations - przedstawia nieopłacone i już anulowane z tego powodu rezerwacje

```
CREATE VIEW V_UnpayedCancelledReservations
AS
SELECT conf.ConferenceName, c.ClientName
FROM Payments p
JOIN ClientReservations cr
ON cr.ClientReservationID = p.PaymentID
JOIN Clients c
ON c.ClientID = cr.ClientID
JOIN Conferences conf
ON conf.ConferenceID = cr.ConferenceID
WHERE p.FineAssessed < p.FinePaid AND cr.IsCancelled = 1
```

GO

V_UnpayedNotCancelledReservations - przedstawia listę nieopłaconych ale wciąż aktywnych rezerwacji

```
CREATE VIEW V_UnpayedNotCancelledReservations
AS
    SELECT conf.ConferenceName, c.ClientName, (p.FineAssessed - p.FinePaid) as Difference
        FROM Payments p
        JOIN ClientReservations cr
        ON cr.ClientReservationID = p.PaymentID
        JOIN Clients c
        ON c.ClientID = cr.ClientID
        JOIN Conferences conf
        ON conf.ConferenceID = cr.ConferenceID
        WHERE p.FineAssessed < p.FinePaid AND cr.IsCancelled = 0
```

GO

V_OverPayedReservations - przedstawia listę wszystkich rezerwacji, które zostały opłacone z nadmiarem

```
CREATE VIEW V_OverPayedReservations
AS
    SELECT conf.ConferenceName, c.ClientName, (p.FinePaid - p.FineAssessed) as Difference
        FROM Payments p
        JOIN ClientReservations cr
        ON cr.ClientReservationID = p.PaymentID
        JOIN Clients c
        ON c.ClientID = cr.ClientID
        JOIN Conferences conf
        ON conf.ConferenceID = cr.ConferenceID
        WHERE p.FinePaid > p.FineAssessed
```

GO

V_PayedReservations - przedstawia listę opłaconych, przed rozpoczęciem konferencji, rezerwacji

```
CREATE VIEW V_PayedReservations
AS
SELECT conf.ConferenceName, c.ClientName
FROM Payments p
JOIN ClientReservations cr
ON cr.ClientReservationID = p.PaymentID
JOIN Clients c
ON c.ClientID = cr.ClientID
JOIN Conferences conf
ON conf.ConferenceID = cr.ConferenceID
WHERE p.FinePaid = p.FineAssessed AND conf.StartDate > CONVERT(date, GETDATE())
GO
```

V_CancelledConferencesReseravtions - przedstawia listę anulowanych rezerwacji na konferencje

```
CREATE VIEW V_CancelledConferencesReseravtions
AS
SELECT c.ClientID, c.ClientName, c.ClientSurname, conf.ConferenceID, conf.ConferenceName
FROM Clients AS c
INNER JOIN ClientReservations AS cr
ON c.ClientID = cr.ClientID
INNER JOIN Conferences AS conf
ON cr.ConferenceID = conf.ConferenceID
WHERE cr.IsCancelled = 1
GO
```

V_ClientsList - przedstawia liste wszystkich klientów, którzy kiedykolwiek rejestrówali się na jakąkolwiek konferencje

```
CREATE VIEW V_ClientsList
AS
SELECT ClientID,
IIF(IsPrivate = 0, ClientName, ClientName + ' ' + ClientSurname) as Name
```

```
,  
    PhoneNumber,  
    Email,  
    Address,  
    City,  
    PostalCode,  
    Country  
FROM Clients
```

```
GO
```

Indeksy

Podczas tworzenia indeksów staraliśmy się wybrać te atrybuty, które bardzo często są wykorzystywane do zanajdowania informacji w procedurach, funkcjach, widokach i triggerach. Są to przeważnie identyfikatory relacji.

```
CREATE INDEX I_Clients ON Clients (ClientID)
```

```
CREATE INDEX I_ClientReservations ON ClientReservations (ClientReservationID)
```

```
CREATE INDEX I_Payments ON Payments (PaymentID)
```

```
CREATE INDEX I_Participants ON Participants (ParticipantID)
```

```
CREATE INDEX I_PriceList ON PriceList (PriceID)
```

```
CREATE INDEX I_Conferences ON Conferences (ConferenceID)
```

```
CREATE INDEX I_Workshops ON Workshops (WorkshopID)
```

```
CREATE INDEX I_DaysReservations ON DaysReservations (DayReservationID)
```

```
CREATE INDEX I_WorkshopsReservations ON WorkshopsReservations (WorkshopReservationID)
```

```
CREATE INDEX I_ParticipantEmail ON Participants (Email)
```

```
CREATE INDEX I_ClientEmail ON Clients (Email)
```

Role

Tworzona przez nas baza danych przewiduje następujące role w systemie:

- Administrator - osoba znająca język SQL, będzie mogła dalej rozwijać stworzoną przez nas bazę danych. Będzie również miała nieograniczone uprawnienia.
- Pracownik - osoba odpowiedzialna za wprowadzanie wszelkich danych do bazy takich jak: konferencje, warsztaty, progi cenowe, rezerwacje, klienci, uczestnicy. Posiada dostęp do wszystkich procedur, funkcji i widoków. Jej zadaniem jest również pomóc klientom i uczestnikom w rezerwacji miejsc w przypadku kiedy wspomniane osoby mają problem z rezerwacją przez internet.
- Klient - będzie posiadał możliwość rezerwacji miejsc na konferencje i warsztaty ich zmiany oraz dodawania uczestników do systemu. Dostęp do:
 - Procedur:
 - AddParticipant
 - AddParticipantForConferenceDay
 - AddParticipantForWorkshop
 - AddReservationForConference
 - AddReservationForConferenceDay
 - AddReservationForWorkshop
 - CancelConferenceReservation
 - CancelParticipantReservation
 - CancelDayReservation
 - CancelParticipantWorkshopReservation
 - CancelWorkshopReservation
 - ChangeDayReservationPlaces
 - ChangeWorkshopReservationPlaces
 - CheckCurrentPayment
 - Funkcji:
 - AllPaymentsByClientID (ClientID to ID Klienta automatycznie przekazywane do funkcji)
 - ClientReservationHistory
 - FreeAndReservedPlacesForConference (tylko dla konferencji, na które klient jest już zarejestrowany)
 - FreeAndReservedPlacesForWorkshop (tylko dla warsztatów odbywających się w ramach konferencji, na które klient jest już zarejestrowany)
 - NonregulatedPaymentsByClientID - (ClientID to ID Klienta automatycznie przekazywane do funkcji)
 - RegulatedPaymentsByClientID - (ClientID to ID Klienta automatycznie przekazywane do funkcji)

- ShowPrices
 - ShowWorkshops
 - GetCurrentPrice
- Uczestnik - będzie posiadał możliwość wpisania się na dany dzień konferencji lub na dany warsztat, a także anulowania tychże wpisów. Dostęp do:
 - Procedur:
 - AddParticipantForConferenceDay
 - AddParticipantForWorkshop
 - CancelParticipantReservation
 - CancelParticipantWorkshopReservation
 - Funkcji:
 - ShowWorkshops
 - FreeAndReservedPlacesForConference
 - FreeAndReservedPlacesForWorkshop

Generator danych

- Dane dla tabel Clients, Participants oraz Conferences zostały wygenerowane przy pomocy strony internetowej <https://mockaroo.com/>
- Dane dla tabeli Workshops generuje skrypt

```
USE mmandows_a
DECLARE @Iterator INT,
@DayIterator INT,
@WorkshopIterator INT,
@NumberOfDays INT,
@ConferenceDay INT,
@WorkshopStart TIME,
@WorkshopEnd TIME,
@Places INT,
@WorkshopFee MONEY
```

```
SET @Iterator = 1
```

```
WHILE @Iterator IN

$$(
    \text{SELECT ConferenceID}
    \text{FROM dbo.Conferences}
)$$

BEGIN
```

```

SET @NumberOfDays = DATEDIFF(day, (
    SELECT StartDate FROM Conferences WHERE ConferenceID = @Iterator
), (
    SELECT EndDate FROM Conferences WHERE ConferenceID = @Iterator
)) + 1

SET @DayIterator = 1

WHILE @DayIterator <= @NumberOfDays
BEGIN

    SET @WorkshopStart = '8:00'
    SET @WorkshopEnd = '10:00'
    SET @WorkshopIterator = 0

    WHILE @WorkshopIterator < 4
    BEGIN

        SET @Places = CONVERT(INT, RAND()*10 + 20)
        SET @WorkshopFee = ROUND( CONVERT(MONEY, RAND()*20 + 20), 2)

        EXEC P_AddWorkshop @ConferenceID = @Iterator,
                            @ConferenceDay = @DayIterator,
                            @WorkshopName = 'A',
                            @Places = @Places,
                            @WorkshopFee = @WorkshopFee,
                            @WorkshopStart = @WorkshopStart,
                            @WorkshopEnd = @WorkshopEnd

        SET @WorkshopStart = DATEADD (minute, 120, @WorkshopStart)
        SET @WorkshopEnd = DATEADD (minute, 120, @WorkshopEnd)
        SET @WorkshopIterator += 1

    END

    SET @DayIterator += 1

END

SET @Iterator += 1

END
GO

```

- Dane dla tabeli PriceList zostały wygenerowane przy użyciu napisanego przez nas krótkiego polecenia SQL, korzystającego z wcześniej uzupełnionych danych w w/w tabelach

```
-- wstawianie danych do PriceList
DECLARE @Iterator      INT
= 1

DECLARE @PriceValue MONEY
= 50

DECLARE @FirstDate     DATE
DECLARE @SecondDate    DATE
DECLARE @ThirdDate     DATE

DECLARE @ConferenceDate DATE

WHILE @Iterator IN
(
    SELECT ConferenceID
    FROM dbo.Conferences
)
BEGIN

    SET @ConferenceDate =
    (
        SELECT StartDate
        FROM dbo.Conferences
        WHERE ConferenceID = @Iterator
    )
    SET @FirstDate  = DATEADD(MONTH, -2, @ConferenceDate)
    SET @SecondDate = DATEADD(MONTH, -1, @ConferenceDate)
    SET @ThirdDate  = DATEADD(MONTH, -0, @ConferenceDate)

    EXEC dbo.P_AddPriceToConferencePriceList @ConferenceID = @Iterator,
-- int
-- money
@PriceValue      = @PriceValue,
@PriceDate       = @FirstDate
```

```

-- date

SET @PriceValue = @PriceValue + 50

EXEC dbo.P_AddPriceToConferencePriceList @ConferenceID = @Iterator,      -- i
nt
@PriceValue = @PriceValue,          -- m
oney
@PriceDate = @SecondDate         -- -
date

SET @PriceValue = @PriceValue + 100

EXEC dbo.P_AddPriceToConferencePriceList @ConferenceID = @Iterator,      -- i
nt
@PriceValue = @PriceValue,          -- m
oney
@PriceDate = @ThirdDate -- date

SET @Iterator = @Iterator + 1

END
GO

```

- Dane do tabeli ClientReservations zostały wygenerowane przy użyciu poniższego krótkiego skryptu

```

DECLARE @i10      INT
= 1

DECLARE @clientID   INT
= 1

DECLARE @confID     INT
= 1

WHILE @confID IN
(
    SELECT ConferenceID
    FROM dbo.Conferences

```

```

)
BEGIN
    WHILE @i10 <= 10
        BEGIN

            DECLARE @ConferenceStartDate      DATE
            = (
                SELECT      StartDate
                FROM        dbo.Conferences
                WHERE       ConferenceID = @confID
            )

            DECLARE @ReservationDate      DATE
            = DATEADD(MONTH, -2, @ConferenceStartDate)

            INSERT INTO dbo.ClientReservations
            (
                ConferenceID,
                ClientID,
                ReservationDate,
                IsCancelled
            )
            VALUES
            (
                @confID,          -- ConferenceID - int
                @clientID,        -- ClientID - int
                @ReservationDate, -- ReservationDate - date
                0 -- IsCancelled - bit
            )

            SET @clientID = @clientID + 1
            SET @i10 = @i10 + 1
        END
    SET @i10 = 1
    SET @confID = @confID + 1
END
GO

```

- Dane do tabeli Payments są generowane automatycznie przy dokonywaniu, anulowaniu lub zmianie ilości zarezerwowanych miejsc. Odpowiedzialna jest za to procedura

- P_CountFine

oraz Triggery:

- T_CountFineAfterConferenceDayReservationOrUpdate
- T_CountFineAfterWorkhopReservationOrUpdate
- T_CountFineAfterConferenceDayReservationOrUpdate

- Dane do tabeli DaysReservations generuje skrypt

```

DECLARE @Iterator INT,
@NumberOfDays INT,
@ConferenceID INT,
@DayIterator INT,
@NormalRes INT,
@studentRes INT

SET @Iterator = 1

WHILE @Iterator IN
(
    SELECT ClientReservationID
    FROM ClientReservations
)
BEGIN

    SET @ConferenceID = (
        SELECT ConferenceID FROM ClientReservations WHERE ClientReservationID = @Iterator
    )

    SET @NumberOfDays = DATEDIFF(day, (
        SELECT StartDate FROM Conferences WHERE ConferenceID = @ConferenceID
    ), (
        SELECT EndDate FROM Conferences WHERE ConferenceID = @ConferenceID
    )) + 1
    SET @DayIterator = 1
    WHILE @DayIterator <= @NumberOfDays
    BEGIN
        SET @NormalRes = CONVERT(INT, RAND()*10 + 20)
        SET @studentRes = CONVERT(INT, RAND()*10 + 20)
    END
END

```

```

EXEC P_AddReservationForConferenceDay
    @ClientReservationID = @Iterator,
    @ConferenceDay = @DayIterator,
    @NormalReservations = @NormalRes,
    @StudentReservations = @StudentRes

SET @DayIterator += 1
END

SET @Iterator += 1
END
GO

```

- Dane do tabeli WorkshopReservations generowane są przez skrypt

```

DECLARE @Iterator INT,
        @WorkshopIterator INT,
        @ConferenceID INT

SET @Iterator = 1

WHILE @Iterator IN
(
    SELECT DayReservationID
    FROM DaysReservations
)
BEGIN

    SET @ConferenceID = (
        SELECT ConferenceID
        FROM ClientReservations cr
        JOIN DaysReservations dr ON dr.ClientReservationID = cr.ClientReservationID
        WHERE @Iterator = dr.DayReservationID
    )

    SET @WorkshopIterator = (
        SELECT TOP 1 WorkshopID
        FROM Workshops
    )

```

```

    WHERE ConferenceID = @ConferenceID
    ORDER BY WorkshopID
)

WHILE @WorkshopIterator IN
(
    SELECT WorkshopID
    FROM Workshops
    WHERE ConferenceID = @ConferenceID
)
BEGIN
    EXEC P_AddReservationForWorkshop
        @DayReservationID = @Iterator,
        @WorkshopID = @WorkshopIterator,
        @NormalReservations = 10

    SET @WorkshopIterator += 1
END

SET @Iterator += 1

END
GO

```

- Dane do tabeli ParticipantReservations generuje skryp

```

DECLARE @PartID      INT
      = 1
DECLARE @DayID       INT
      = 1
DECLARE @NormalRes   INT
      = 0
DECLARE @StudentRes  INT
      = 0
DECLARE @StudentCard INT

WHILE @DayID IN
(
    SELECT DayReservationID

```

```

        FROM dbo.DaysReservations
    )
BEGIN

    SET @NormalRes =
    (
        SELECT NormalReservations
        FROM dbo.DaysReservations
        WHERE DayReservationID = @DayID
    )
    SET @StudentRes =
    (
        SELECT StudentsReservations
        FROM dbo.DaysReservations
        WHERE DayReservationID = @DayID
    )

    WHILE @NormalRes > 0
    BEGIN

        IF @PartID > 1000
        BEGIN
            SET @PartID = 1
        END

        EXEC dbo.P_AddParticipantForConferenceDay @ParticipantID = @PartID,
-- int
-- int
-- int
-- int
-- te
            @DayReservationID = @DayID,
            @StudentCard = NULL,
            @StudentCardDate = NULL -- da
        SET @PartID = @PartID +1
        SET @NormalRes = @NormalRes - 1

    END

    WHILE @StudentRes> 0
    BEGIN

```

```

IF @PartID > 1000
BEGIN
    SET @PartID = 1
END

SET @StudentCard = RAND()*1000000
EXEC dbo.P_AddParticipantForConferenceDay @ParticipantID = @PartID,
-- int
-- int
-- int
-- int
@DayReservationID = @DayID,
@StudentCard = @StudentCard,
@StudentCardDate = '2018-05-1
7' -- date
SET @PartID = @PartID +1
SET @StudentRes = @StudentRes - 1

END

SET @DayID = @DayID + 1

END
GO

```

- Dane do tabeli ParticipantWorkshops generuje skrypt

```

DECLARE @ParticipantReservationID1 INT,
        @ParticipantReservationID2 INT,
        @WorkshopID INT,
        @DayReservationID INT

SET @WorkshopID = 1

WHILE @WorkshopID IN
(
    SELECT WorkshopID
    FROM Workshops

```

```
)  
BEGIN  
  
SET @DayReservationID = (  
    SELECT TOP 1 DayReservationID  
    FROM WorkshopsReservations  
    WHERE WorkshopID = @WorkshopID  
    ORDER BY DayReservationID  
)  
  
WHILE @DayReservationID IN (  
    SELECT DayReservationID  
    FROM WorkshopsReservations  
    WHERE WorkshopID = @WorkshopID  
)  
BEGIN  
    SET @ParticipantReservationID1 = (  
        SELECT TOP 1 ParticipantReservationID  
        FROM ParticipantReservations  
        WHERE DayReservationID = @DayReservationID  
        ORDER BY ParticipantReservationID  
)  
  
    SET @ParticipantReservationID2 = (  
        SELECT TOP 1 ParticipantReservationID  
        FROM ParticipantReservations  
        WHERE DayReservationID = @DayReservationID  
        ORDER BY ParticipantReservationID DESC  
)  
  
    EXEC P_AddParticipantForWorkshop  
        @ParticipantReservationID = @ParticipantReservationID1,  
        @WorkshopID = @WorkshopID  
  
    EXEC P_AddParticipantForWorkshop  
        @ParticipantReservationID = @ParticipantReservationID2,  
        @WorkshopID = @WorkshopID  
  
    SET @DayReservationID += 1
```

END

SET @workshopID += 1

END

GO