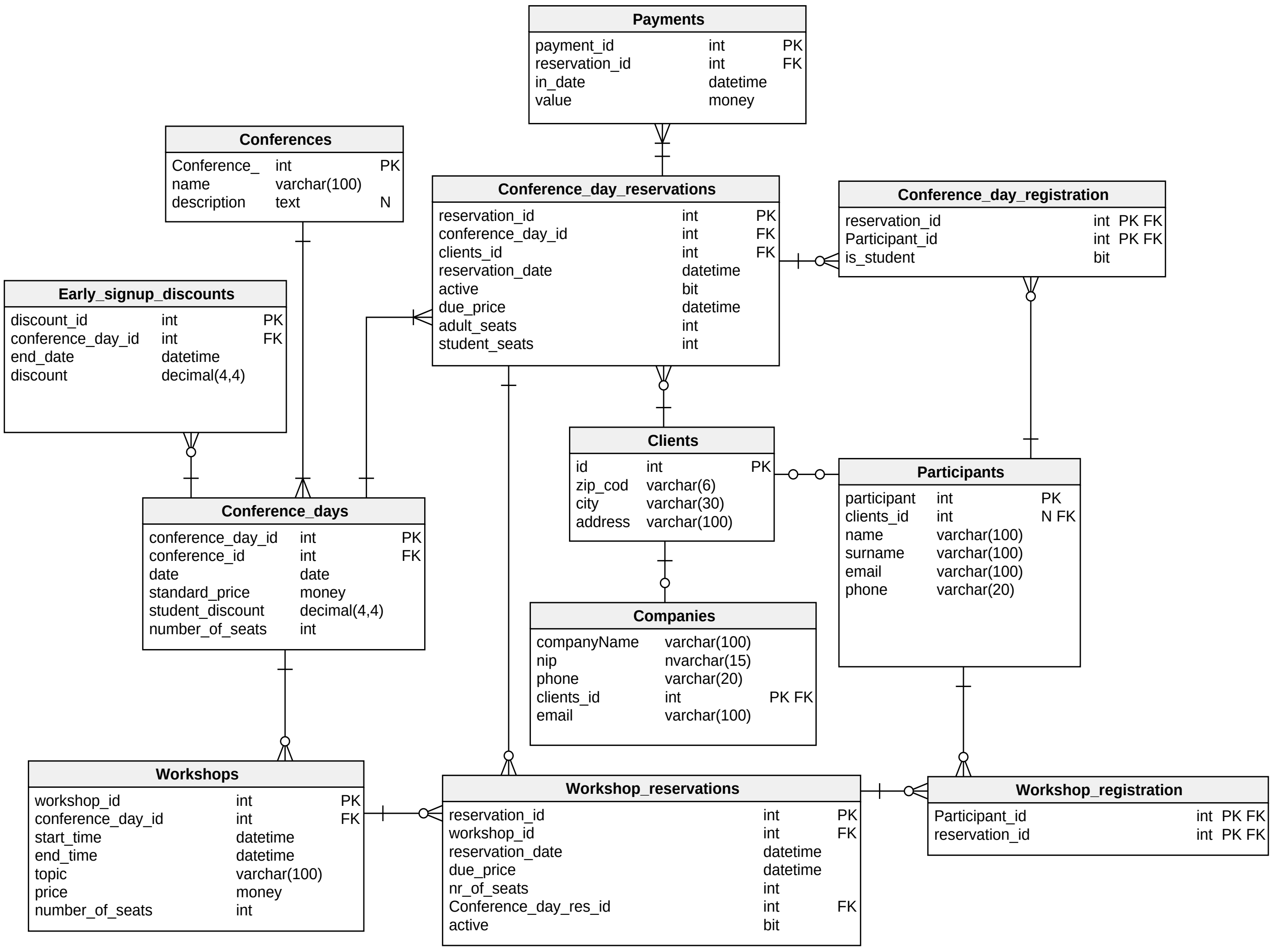
Podstawy baz danych
Projekt konferencje

Agnieszka Dutka, Maciek Trątnowiecki

AGH, Styczeń 2020

Objaśnienie schematu bazy

- Clients - Reprezentuje klientów chcących opłacić miejsca na konferencjach i warsztatach. Klientem może być zarówno firma, jak i osoba prywatna. W zależności od tego dane klienta reprezentowane są przez odpowiednią relację w bazie.
- Companies - Jeśli klient jest firmą, przechowuje jego dane.
- Participants - Jeśli klient jest osobą prywatną, przechowuje jego dane.
- Conferences - Reprezentuje konferencję z którą powiązane są odpowiednie dni konferencyjne, oraz warsztaty.
- Conference_days - Reprezentuje pojedynczy dzień konferencji. Powiązana jest z nim ustalona opłata za uczestnictwo. Zniżki obowiązujące w zależności od daty rejestracji zwarte są w relacji Early_Signup_Discounts.
- Early_Signup_Discounts - Odpowiada za informację o tabeli zniżek na dany dzień konferencyjny. Pojedyncza zniżka przechowywana jest w krotce z atrybutami w postaci procentowej obniżki ceny standardowej, oraz ostatniego dnia w którym obowiązuje.
- Conference_day_reservations - Realizuje rezerwacje na poszczególny dzień konferencji. Każda rezerwacja powiązana jest z klientem, który ją opłaca. Za powiązanie rezerwacji z uczestnikiem odpowiada osobna relacja. Zawiera także pole due_price określające termin płatności. Atrybut active odpowiada za możliwość rezygnacji z podjętej rezerwacji (uznaliśmy, że usuwanie krotki z bazy może nie być optymalnym rozwiązaniem, jako że zawarte w niej dane mogą jeszcze być przydatne z punktu widzenia logiki biznesowej). Atrybuty adult_seats i student_seats służą do liczenia kosztu podjęcia rezerwacji przed powiązaniem jej z uczestnikami konferencji.
- Conference_day_registration - Wiąże rezerwację z uczestnikami konferencji. Atrybut is_student informuje, czy danemu uczestnikowi przysługuje zniżka studencka.
- Payments - Przechowuje informacje o wpływach pieniężnych powiązanych z daną rejestracją.
- Workshops - Reprezentuje warsztaty odbywające się w trakcie odpowiednich dni konferencyjnych.
- Workshops_reservations - Opisuje rezerwacje na warsztaty w sposób analogiczny do rezerwacji na konferencje.
- Workshops_registrations - Łączy rezerwację z uczestnikami w sposób analogiczny do dni konferencyjnych.



Payments

payment_id	int	PK
reservation_id	int	FK
in_date	datetime	
value	money	

Conferences

Conference_	int	PK
name	varchar(100)	
description	text	N

Conference_day_reservations

reservation_id	int	PK
conference_day_id	int	FK
clients_id	int	FK
reservation_date	datetime	
active	bit	
due_price	datetime	
adult_seats	int	
student_seats	int	

Conference_day_registration

reservation_id	int	PK FK
Participant_id	int	PK FK
is_student	bit	

Early_signup_discounts

discount_id	int	PK
conference_day_id	int	FK
end_date	datetime	
discount	decimal(4,4)	

Clients

id	int	PK
zip_cod	varchar(6)	
city	varchar(30)	
address	varchar(100)	

Participants

participant	int	PK
clients_id	int	N FK
name	varchar(100)	
surname	varchar(100)	
email	varchar(100)	
phone	varchar(20)	

Conference_days

conference_day_id	int	PK
conference_id	int	FK
date	date	
standard_price	money	
student_discount	decimal(4,4)	
number_of_seats	int	

Companies

companyName	varchar(100)	
nip	nvarchar(15)	
phone	varchar(20)	
clients_id	int	PK FK
email	varchar(100)	

Workshops

workshop_id	int	PK
conference_day_id	int	FK
start_time	datetime	
end_time	datetime	
topic	varchar(100)	
price	money	
number_of_seats	int	

Workshop_reservations

reservation_id	int	PK
workshop_id	int	FK
reservation_date	datetime	
due_price	datetime	
nr_of_seats	int	
Conference_day_res_id	int	FK
active	bit	

Workshop_registration

Participant_id	int	PK FK
reservation_id	int	PK FK

Implementacja

```sql
-- tables
-- Table: Clients
CREATE TABLE Clients (
    id int  NOT NULL IDENTITY,
    zip_code varchar(6)  NOT NULL,
    city varchar(30)  NOT NULL,
    address varchar(100)  NOT NULL,
    CONSTRAINT Clients_pk PRIMARY KEY  (id)
);

-- Table: Companies
CREATE TABLE Companies (
    companyName varchar(100)  NOT NULL,
    nip nvarchar(15)  NOT NULL CHECK ((nip not like '%[^0-9]%') and (LEN(nip) = 10) and (nip
    not like '0%' or nip like '1%')),
    phone varchar(20)  NOT NULL,
    clients_id int  NOT NULL,
    email varchar(100)  NOT NULL CHECK (email like '%_@__%.__%'),
    CONSTRAINT unique_nip UNIQUE (nip),
    CONSTRAINT checkNip CHECK (dbo.IsValidNip(nip) = 1),
    CONSTRAINT Companies_pk PRIMARY KEY  (clients_id)
);

CREATE INDEX companies_client_id on Companies (clients_id ASC)
;

CREATE INDEX companies_nip on Companies (nip ASC)
;

-- Table: Conference_day_registration
CREATE TABLE Conference_day_registration (
    reservation_id int  NOT NULL,
    Participant_id int  NOT NULL,
    is_student bit  NOT NULL DEFAULT 0,
    CONSTRAINT Conference_day_registration_pk PRIMARY KEY  (reservation_id,Participant_id)
);

CREATE INDEX day_reg_part_id on Conference_day_registration (Participant_id ASC)
;

CREATE INDEX day_reg_res_id on Conference_day_registration (reservation_id ASC)
;

-- Table: Conference_day_reservations
CREATE TABLE Conference_day_reservations (
    reservation_id int  NOT NULL IDENTITY,
    conference_day_id int  NOT NULL,
    clients_id int  NOT NULL,
    reservation_date datetime  NOT NULL DEFAULT GETDATE(),
    active bit  NOT NULL DEFAULT 1,
    due_price datetime  NOT NULL DEFAULT DATEADD(week, 2, GETDATE()) CHECK (due_price >=
    GETDATE()),
    adult_seats int  NOT NULL DEFAULT 0 CHECK (adult_seats >= 0),
    student_seats int  NOT NULL DEFAULT 0 CHECK (student_seats >= 0),
    CONSTRAINT Conference_day_reservations_pk PRIMARY KEY  (reservation_id)
);

CREATE INDEX day_res_day_id on Conference_day_reservations (conference_day_id ASC)
;

CREATE INDEX day_res_client_id on Conference_day_reservations (clients_id ASC)
;

CREATE INDEX day_res_pay_deadline on Conference_day_reservations (due_price ASC)
;

-- Table: Conference_days
CREATE TABLE Conference_days (
    conference_day_id int  NOT NULL IDENTITY,
    conference_id int  NOT NULL,
    date date  NOT NULL DEFAULT GETDATE(),
    standard_price money  NOT NULL DEFAULT 0 CHECK (standard_price >= 0),
    student_discount decimal(4,4)  NOT NULL DEFAULT 0 CHECK (student_discount >= 0),
    number_of_seats int  NOT NULL DEFAULT 0 CHECK (number_of_seats >= 0),
    CONSTRAINT Conference_days_pk PRIMARY KEY  (conference_day_id)
);

CREATE INDEX day_conf_id on Conference_days (conference_id ASC)
;

-- Table: Conferences
CREATE TABLE Conferences (
     Conference_id int  NOT NULL IDENTITY,
    name varchar(100)  NOT NULL,
     description text  NULL,
    CONSTRAINT Conferences_pk PRIMARY KEY  (Conference_id)
);

-- Table: Early_signup_discounts
CREATE TABLE Early_signup_discounts (
     discount_id int  NOT NULL IDENTITY,
     conference_day_id int  NOT NULL,
     end_date datetime  NOT NULL,
     discount decimal(4,4)  NOT NULL DEFAULT 0,
    CONSTRAINT Early_signup_discounts_pk PRIMARY KEY  (discount_id)
);

CREATE INDEX esd_end_date on Early_signup_discounts (end_date ASC)
;

-- Table: Participants
CREATE TABLE Participants (
     participant_id int  NOT NULL IDENTITY,
     clients_id int  NULL DEFAULT Null,
    name varchar(100)  NOT NULL,
    surname varchar(100)  NOT NULL,
     email varchar(100)  NOT NULL CHECK (email like '%_@__%.__%'),
     phone varchar(20)  NOT NULL,
    CONSTRAINT Participants_pk PRIMARY KEY  (participant_id)
);

CREATE INDEX participants_client_id on Participants (clients_id ASC)
;

CREATE INDEX participants_name on Participants (surname ASC,name ASC)
;

-- Table: Payments
CREATE TABLE Payments (
     payment_id int  NOT NULL IDENTITY,
     reservation_id int  NOT NULL,
     in_date datetime  NOT NULL,
     value money  NOT NULL,
    CONSTRAINT Payments_pk PRIMARY KEY  (payment_id)
);

CREATE INDEX payment_reservation on Payments (reservation_id ASC)
;

-- Table: Workshop_registration
CREATE TABLE Workshop_registration (
     Participant_id int  NOT NULL,
     reservation_id int  NOT NULL,
    CONSTRAINT Workshop_registration_pk PRIMARY KEY  (Participant_id,reservation_id)
);

CREATE INDEX workshop_reg_part_id on Workshop_registration (Participant_id ASC)
;

CREATE INDEX workshop_reg_res_id on Workshop_registration (reservation_id ASC)
;

-- Table: Workshop_reservations
CREATE TABLE Workshop_reservations (
     reservation_id int  NOT NULL IDENTITY,
     workshop_id int  NOT NULL,
     reservation_date datetime  NOT NULL DEFAULT GETDATE(),
     due_price datetime  NOT NULL DEFAULT DATEADD(week, 2, GETDATE()) CHECK (due_price >=
    GETDATE()),
     nr_of_seats int  NOT NULL DEFAULT 0 CHECK (nr_of_seats >= 0),
    Conference_day_res_id int  NOT NULL,
    active bit  NOT NULL DEFAULT 1,
    CONSTRAINT Workshop_reservations_pk PRIMARY KEY  (reservation_id)
);

CREATE INDEX workshop_res_conf_res_id on Workshop_reservations (Conference_day_res_id ASC)
;

CREATE INDEX workshop_res_pay_deadline on Workshop_reservations (due_price ASC)
;

CREATE INDEX workshop_res_workshop_id on Workshop_reservations (workshop_id ASC)
;

-- Table: Workshops
CREATE TABLE Workshops (
    workshop_id int  NOT NULL IDENTITY,
    conference_day_id int  NOT NULL,
    start_time datetime  NOT NULL,
    end_time datetime  NOT NULL CHECK (end_time >= GETDATE()),
    topic varchar(100)  NOT NULL,
    price money  NOT NULL CHECK (price >= 0),
    number_of_seats int  NOT NULL DEFAULT 0 CHECK (number_of_seats >= 0),
    CONSTRAINT Workshops_pk PRIMARY KEY  (workshop_id)
);

CREATE INDEX workshop_day_id on Workshops (conference_day_id ASC)
;

CREATE INDEX workshop_time on Workshops (start_time ASC,end_time ASC)
;

-- foreign keys
-- Reference: Companies_Clients (table: Companies)
ALTER TABLE Companies ADD CONSTRAINT Companies_Clients
    FOREIGN KEY (clients_id)
    REFERENCES Clients (id);

-- Reference: Conference_day_registration_Conference_day_reservations (table:
    Conference_day_registration)
ALTER TABLE Conference_day_registration ADD CONSTRAINT
    Conference_day_registration_Conference_day_reservations
    FOREIGN KEY (reservation_id)
    REFERENCES Conference_day_reservations (reservation_id)
    ON DELETE  CASCADE;

-- Reference: Conference_day_registration_Participants (table: Conference_day_registration)
ALTER TABLE Conference_day_registration ADD CONSTRAINT
    Conference_day_registration_Participants
    FOREIGN KEY (Participant_id)
    REFERENCES Participants (participant_id)
    ON DELETE  CASCADE;

-- Reference: Conference_day_reservations_Clients (table: Conference_day_reservations)
ALTER TABLE Conference_day_reservations ADD CONSTRAINT Conference_day_reservations_Clients
    FOREIGN KEY (clients_id)
    REFERENCES Clients (id);

-- Reference: Conference_day_reservations_Conference_days (table: Conference_day_reservations)
ALTER TABLE Conference_day_reservations ADD CONSTRAINT
    Conference_day_reservations_Conference_days
    FOREIGN KEY (conference_day_id)
    REFERENCES Conference_days (conference_day_id)
    ON DELETE  CASCADE;

-- Reference: Conference_days_Conferences (table: Conference_days)
ALTER TABLE Conference_days ADD CONSTRAINT Conference_days_Conferences
    FOREIGN KEY (conference_id)
    REFERENCES Conferences (Conference_id)
    ON DELETE  CASCADE;

-- Reference: Discounts_Conference_days (table: Early_signup_discounts)
ALTER TABLE Early_signup_discounts ADD CONSTRAINT Discounts_Conference_days
    FOREIGN KEY (conference_day_id)
    REFERENCES Conference_days (conference_day_id)
    ON DELETE  CASCADE;
```

```
-- Reference: Participants_Clients (table: Participants)
ALTER TABLE Participants ADD CONSTRAINT Participants_Clients
    FOREIGN KEY (clients_id)
    REFERENCES Clients (id)
    ON DELETE  SET NULL;

-- Reference: Payments_Conference_day_reservations (table: Payments)
ALTER TABLE Payments ADD CONSTRAINT Payments_Conference_day_reservations
    FOREIGN KEY (reservation_id)
    REFERENCES Conference_day_reservations (reservation_id);

-- Reference: Workshop_registration_Participants (table: Workshop_registration)
ALTER TABLE Workshop_registration ADD CONSTRAINT Workshop_registration_Participants
    FOREIGN KEY (Participant_id)
    REFERENCES Participants (participant_id)
    ON DELETE  CASCADE;

-- Reference: Workshop_registration_Workshop_reservations (table: Workshop_registration)
ALTER TABLE Workshop_registration ADD CONSTRAINT Workshop_registration_Workshop_reservations
    FOREIGN KEY (reservation_id)
    REFERENCES Workshop_reservations (reservation_id)
    ON DELETE  CASCADE;

-- Reference: Workshop_reservations_Conference_day_reservations (table: Workshop_reservations)
ALTER TABLE Workshop_reservations ADD CONSTRAINT
     Workshop_reservations_Conference_day_reservations
    FOREIGN KEY (Conference_day_res_id)
    REFERENCES Conference_day_reservations (reservation_id)
    ON DELETE  CASCADE;

-- Reference: Workshop_reservations_Workshops (table: Workshop_reservations)
ALTER TABLE Workshop_reservations ADD CONSTRAINT Workshop_reservations_Workshops
    FOREIGN KEY (workshop_id)
    REFERENCES Workshops (workshop_id)
    ON DELETE  CASCADE;

-- Reference: Workshops_Conference_days (table: Workshops)
ALTER TABLE Workshops ADD CONSTRAINT Workshops_Conference_days
    FOREIGN KEY (conference_day_id)
    REFERENCES Conference_days (conference_day_id);

-- End of file.
```

../Create.sql

Widoki

```
-- Shows sum of payments per reservation
CREATE VIEW PaymentsForReservation
AS
SELECT reservation_id, ISNULL(sum(Payments.value), 0) as already_paid
FROM Payments
GROUP BY reservation_id
GO

-- Show sum of payments per client
CREATE VIEW PaymentsForClients
AS
SELECT clients_id, SUM(value) AS paid
FROM Payments
        INNER JOIN Conference_day_reservations Cdr ON Payments.reservation_id = Cdr.
    reservation_id
GROUP BY clients_id
GO


-- Shows balance of payments per reservation
CREATE VIEW BalanceForReservation
AS
SELECT reservation_id, already_paid - dbo.confReservationPrice(reservation_id) as balance
FROM PaymentsForReservation
GO

-- Shows payment balance per client
CREATE VIEW ClientsBalanceView
AS
SELECT id                    as client_id,
       dbo.clientsBalance(id) as balance
FROM Clients
GO

-- Shows clients with unpaid reservations
CREATE VIEW ClientsWithUnpaidReservations
AS
SELECT clients_id,
       reservation_id,
       due_price
      as to_pay_until,
       DATEDIFF(DAY, GETDATE(), due_price) as days_left,
       dbo.confReservationPrice(reservation_id) - dbo.confReservationPaidAmount(reservation_id
    ) as left_to_pay,
       dbo.showClientsName(clients_id)
      as clients_name
FROM Conference_day_reservations
WHERE dbo.confReservationPrice(reservation_id) - dbo.confReservationPaidAmount(reservation_id)
     > 0.01
  and Conference_day_reservations.active = 1
GO

-- Same but only for companies
CREATE VIEW CompaniesWithUnpaidReservations
AS
    SELECT *
    FROM ClientsWithUnpaidReservations
    WHERE dbo.isClientCompany(clients_id) = 1
GO

-- Same but only for individual clients
CREATE VIEW IndividualClientsWithUnpaidReservations
AS
    SELECT *
    FROM ClientsWithUnpaidReservations
    WHERE dbo.isClientCompany(clients_id) = 0
GO

-- Same, but with the earliest payment deadline
CREATE VIEW UnpaidReservationsWithEarliestDeadline
AS
    SELECT TOP 10 *
    FROM ClientsWithUnpaidReservations
    ORDER BY to_pay_until ASC
```

```
GO

-- Clients with unpaid reservations who exceeded payment deadline
CREATE VIEW ClientsWithExceededPaymentDeadline
AS
    SELECT *
    FROM ClientsWithUnpaidReservations
    WHERE to_pay_until < GETDATE()
GO

-- Workshops with free seats available
CREATE VIEW WorkshopsWithFreeSeats
AS
SELECT workshop_id,
       dbo.workshopFreeSeats(workshop_id) as free_seats
FROM Workshops
WHERE dbo.workshopFreeSeats(workshop_id) > 0
GO

-- Conference days with free seats available
CREATE VIEW ConferenceDaysWithFreeSeats
AS
SELECT conference_day_id,
       dbo.conferenceFreeSeats(conference_day_id) as free_seats
FROM Conference_days
WHERE dbo.conferenceFreeSeats(conference_day_id) > 0
GO

-- Clients who paid the most
CREATE VIEW ClientsWhoPaidMost
AS
SELECT TOP 20 *
from PaymentsForClients
ORDER BY paid DESC
GO

-- Clients with the highest count of conference reservations
CREATE VIEW MostActiveClients
AS
SELECT TOP 20 clients_id, COUNT(reservation_id) AS reservation_count
FROM Clients
        INNER JOIN Conference_day_reservations Cdr on Clients.id = Cdr.clients_id
GROUP BY clients_id
ORDER BY COUNT(reservation_id) DESC
GO

-- Show participants for conference days
CREATE VIEW ParticipantsForConferenceDay
AS
SELECT name, surname, conference_day_id
FROM Participants
        INNER JOIN Conference_day_registration Cdrg on Participants.participant_id = Cdrg.
    Participant_id
        INNER JOIN Conference_day_reservations Cdr on Cdrg.reservation_id = Cdr.
    reservation_id

-- Show participants for workshops
CREATE VIEW ParticipantsForWorkshop
AS
SELECT name, surname, workshop_id
FROM Participants
        INNER JOIN Workshop_registration Wrg on Participants.participant_id = Wrg.
    Participant_id
        INNER JOIN Workshop_reservations Wrs on Wrg.reservation_id = Wrs.reservation_id

-- Show cancelled conference reservations
CREATE VIEW CancelledConferenceReservations
AS
SELECT reservation_id,
       conference_day_id,
       reservation_date,
       clients_id,
       adult_seats,
       student_seats,
       dbo.showClientsName(clients_id) as clients_name
FROM Conference_day_reservations
```

```
WHERE active = 0

-- Show cancelled workshops reservations
CREATE VIEW CancelledWorkshopsReservations
AS
     SELECT Workshop_reservations.reservation_id,
         workshop_id,
         Workshop_reservations.reservation_date,
         nr_of_seats,
         dbo.showClientsName(clients_id) as clients_name
     FROM Workshop_reservations
           INNER JOIN Conference_day_reservations Cdr on Workshop_reservations.
     Conference_day_res_id = Cdr.reservation_id
     WHERE Workshop_reservations.active = 0
GO

-- Show how popular conferences are
CREATE VIEW ConferencePopularity
AS
     SELECT conf.Conference_id, conf.name, SUM(cdr.student_seats + cdr.adult_seats) AS
     participants_count
     FROM Conferences conf INNER JOIN Conference_days cd on conf.Conference_id = cd.
     conference_id
     INNER JOIN Conference_day_reservations cdr on cd.conference_day_id = cdr.conference_day_id
     WHERE cdr.active = 1
     GROUP BY conf.Conference_id, conf.name
GO

-- Show the most popular conferences
CREATE VIEW MostPopularConference
AS
     SELECT TOP 20 *
     FROM ConferencePopularity
     ORDER BY participants_count DESC
GO

-- Show how popular Workshops are
CREATE VIEW WorkshopPopularity
AS
     SELECT w.workshop_id, w.topic, SUM(wr.nr_of_seats) AS participants_count
     FROM Workshops w INNER JOIN Workshop_reservations wr on w.workshop_id = wr.workshop_id
     WHERE wr.active=1
     GROUP BY w.workshop_id, w.topic
GO

-- Show the most popular workshops
CREATE VIEW MostPopularWorkshops
AS
     SELECT TOP 20 *
     FROM WorkshopPopularity
     ORDER BY participants_count DESC
GO

-- Show all cities from which our clients are coming
CREATE VIEW ClientsCities
AS
     SELECT DISTINCT city
     FROM Clients
GO

-- Show all clients phone numbers
CREATE VIEW ClientsPhones
AS
     SELECT DISTINCT phone
     FROM Clients INNER JOIN Companies C on Clients.id = C.clients_id
     UNION
     SELECT DISTINCT phone
     FROM Clients INNER JOIN Participants P on Clients.id = P.clients_id
GO

-- Show all clients phone numbers
CREATE VIEW ClientsEmails
AS
     SELECT DISTINCT email
     FROM Clients INNER JOIN Companies C on Clients.id = C.clients_id
     UNION
```

```
    SELECT DISTINCT email
    FROM Clients INNER JOIN Participants P on Clients.id = P.clients_id
GO
```

../Views/Views.sql

Funkcje

Funkcja sprawdzająca czy wprowadzony NIP jest zgodny z obowiązującymi prawami.

```
CREATE FUNCTION IsValidNip(
      @nip nvarchar(15)
)
      RETURNS bit
AS
BEGIN
      IF ISNUMERIC(@nip) = 0
            BEGIN
                  RETURN 0
            END

      IF @nip = '0000000000'
            BEGIN
                  RETURN 0
            END
      IF @nip = '1234567891'
            BEGIN
                  RETURN 0
            END
      IF @nip = '1111111111'
            BEGIN
                  RETURN 0
            END
      IF @nip = '1111111112'
            BEGIN
                  RETURN 0
            END
      IF @nip = '9999999999'
            BEGIN
                  RETURN 0
            END
      IF @nip = '1111111112'
            BEGIN
                  RETURN 0
            END

      DECLARE @sum INT;
      SET @sum = 6 * CONVERT(INT, SUBSTRING(@nip, 1, 1)) +
                 5 * CONVERT(INT, SUBSTRING(@nip, 2, 1)) +
                 7 * CONVERT(INT, SUBSTRING(@nip, 3, 1)) +
                 2 * CONVERT(INT, SUBSTRING(@nip, 4, 1)) +
                 3 * CONVERT(INT, SUBSTRING(@nip, 5, 1)) +
                 4 * CONVERT(INT, SUBSTRING(@nip, 6, 1)) +
                 5 * CONVERT(INT, SUBSTRING(@nip, 7, 1)) +
                 6 * CONVERT(INT, SUBSTRING(@nip, 8, 1)) +
                 7 * CONVERT(INT, SUBSTRING(@nip, 9, 1));

      IF CONVERT(TINYINT, SUBSTRING(@nip, 10, 1)) = (@sum % 11)
            BEGIN
                  RETURN 1
            END
      RETURN 0
END
```

../Functions/isValidNip.sql

Funkcja sprawdzająca czy dany klient jest firmą.

```
CREATE FUNCTION isClientCompany(
      @client_id int
)
      RETURNS bit
AS
BEGIN
      DECLARE @res bit = ISNULL((SELECT 1 FROM Companies WHERE clients_id=@client_id),0)
      RETURN @res
END
```

../Functions/isClientCompany.sql

Funkcja zwracająca imię i nazwisko klienta (jeśli jest on osobą fizyczną).

```
CREATE FUNCTION showClientsName(
```

```sql
    @client_id int
)
    RETURNS varchar(100)
AS
BEGIN
    DECLARE @name VARCHAR(100) = (SELECT name + ' ' + surname FROM Participants WHERE
    clients_id = @client_id)
    if @name is not null
        BEGIN
            RETURN @name
        END
    SET @name = (SELECT companyName FROM Companies WHERE clients_id = @client_id)
    RETURN @name
END
```

../Functions/showClientsName.sql

Funkcja zwracająca ilość wolnych miejsc na danej konferencji.

```sql
CREATE FUNCTION conferenceFreeSeats(@conferenceId int)
    RETURNS INT
AS
BEGIN
    DECLARE @used INT
    SET @used = ISNULL((SELECT SUM(adult_seats + student_seats)
                        FROM Conference_day_reservations
                        WHERE conference_day_id = @conferenceId), 0)
    DECLARE @all INT
    SET @all = (SELECT number_of_seats
                FROM Conference_days
                WHERE conference_day_id = @conferenceId)
    RETURN @all - @used
END
GO
```

../Functions/FreeSeats/conferenceFreeSeats.sql

Funkcja zwracająca ilość wolnych miejsc na danych warsztatach.

```sql
CREATE FUNCTION workshopFreeSeats(@workshopId int)
    RETURNS INT
AS
BEGIN
    DECLARE @used INT
    SET @used = ISNULL((SELECT SUM(nr_of_seats)
                        FROM Workshop_reservations
                        WHERE workshop_id = @workshopId), 0)
    DECLARE @all INT
    SET @all = (SELECT number_of_seats
                FROM Workshops
                WHERE workshop_id = @workshopId)
    RETURN @all - @used
END
GO
```

../Functions/FreeSeats/workshopFreeSeats.sql

Funkcja zwracająca imiona i nazwiska uczestników danej konferencji.

```sql
CREATE FUNCTION participantsForGivenConference(
    @conferenceID int
)
    RETURNS table
        AS
        RETURN SELECT name, surname
               FROM ParticipantsForConferenceDay
               WHERE conference_day_id = @conferenceID
```

../Functions/ParticipantsLists/participantsForGivenConference.sql

Funkcja zwracająca imiona i nazwiska uczestników danego warsztatu.

```sql
CREATE FUNCTION participantsForGivenWorkshop(
    @workshopID int
)
    RETURNS table
        AS
        RETURN SELECT name, surname
               FROM ParticipantsForWorkshop
               WHERE workshop_id = @workshopID
```

../Functions/ParticipantsLists/participantsForGivenWorkshop.sql

Funkcja zwracająca całkowity balans danego klienta.

```
CREATE FUNCTION clientsBalance(@ClientId int)
    RETURNS MONEY
AS
BEGIN
    DECLARE @paid MONEY = (SELECT SUM(P.VALUE)
                           FROM Payments P
                                    inner join Conference_day_reservations Cdr on P.
    reservation_id = Cdr.reservation_id
                           WHERE Cdr.clients_id = @ClientId and cdr.active = 1)
    DECLARE @owed MONEY = (SELECT SUM(dbo.confReservationPrice(cdr.reservation_id))
                           FROM Conference_day_reservations cdr
                           WHERE cdr.clients_id = @ClientId and cdr.active = 1)
    DECLARE @res MONEY = ROUND(ISNULL(@paid, 0) - ISNULL(@owed, 0), 2);
    IF @res = 0.01
        BEGIN
            return 0
        END
    RETURN @res
END
```

../Functions/PaymentsAndPrices/clientsBalance.sql

Funkcja zwracająca koszt danej rezerwacji na dzień konferencji (łącznie z ceną podpiętych warsztatów).

```
CREATE FUNCTION confDayPrice(@reservationId int)
    RETURNS MONEY
AS
BEGIN
    DECLARE @earlySignupDiscount decimal(4, 4) = ISNULL((Select TOP 1 esd.discount
                                                          from Conference_day_reservations as
    res
                                                                   inner join Conference_days
    as day on res.conference_day_id = day.conference_day_id
                                                                   inner join
    Early_signup_discounts esd
                                                                              on day.
    conference_day_id = esd.conference_day_id
                                                          where reservation_id = @reservationId
                                                            and esd.end_date > res.
    reservation_date
                                                          order by esd.end_date ASC), 0)
    DECLARE @nrOfStudents int = (SELECT student_seats
                                 FROM Conference_day_reservations
                                 WHERE reservation_id = @reservationId)
    DECLARE @nrOfAdults int = (SELECT adult_seats
                               FROM Conference_day_reservations
                               WHERE reservation_id = @reservationId)
    DECLARE @conferenceDayID int = (SELECT conference_day_id
                                    FROM Conference_day_reservations
                                    WHERE reservation_id = @reservationId)
    DECLARE @standardPrice money = (SELECT standard_price
                                    FROM Conference_days
                                    WHERE conference_day_id = @conferenceDayID)
    DECLARE @studentDiscount decimal(4, 4) = (SELECT student_discount
                                              FROM Conference_days
                                              WHERE conference_day_id = @conferenceDayID)
    DECLARE @price money = (@nrOfAdults * @standardPrice) +
                           (@nrOfStudents * (@standardPrice * (1.0 - @studentDiscount)))
    RETURN CONVERT(money, ROUND(@price * (1.0 - @earlySignupDiscount), 2))
END
GO
```

../Functions/PaymentsAndPrices/confDayPrice.sql

Funkcja zwracająca dotychczasowo wpłaconą kwotę na daną rezerwację.

```
CREATE FUNCTION confReservationPaidAmount(@reservation_id int)
    RETURNS MONEY
AS
```

```
BEGIN
    RETURN ISNULL((SELECT sum(value) FROM Payments WHERE reservation_id = @reservation_id), 0)
END
```
../Functions/PaymentsAndPrices/confReservationPaidAmount.sql

Funkcja zwracająca koszt danej rezerwacji na warsztaty.

```
CREATE FUNCTION confReservationPrice(@reservation_id int)
    RETURNS MONEY
AS
BEGIN
    DECLARE @dayPrice money = dbo.confDayPrice(@reservation_id);
    DECLARE @workshopsPrice money = (select sum(wr.nr_of_seats * W.price)
                                      from Workshop_reservations wr
                                               inner join Workshops W on wr.workshop_id = W.
    workshop_id
                                      where wr.Conference_day_res_id = @reservation_id);

    RETURN ROUND(@dayPrice + ISNULL(@workshopsPrice, 0), 2)
END
```
../Functions/PaymentsAndPrices/confReservationPrice.sql

Procedury

Procedury służące wprowadzaniu nowych danych do bazy:

```
-- Add client to database
CREATE PROCEDURE AddClientParticipant @ZipCode varchar(6), @City varchar(30), @Address varchar
    (100), @Name varchar(100),
                                        @Surname varchar(100),
                                        @Email varchar(100), @Phone varchar(20)
AS
BEGIN
    SET NOCOUNT ON;
    INSERT INTO Clients(zip_code, city, address)
    VALUES (@ZipCode, @City, @Address)

    INSERT INTO Participants (clients_id, name, surname, email, phone)
    VALUES (SCOPE_IDENTITY(), @Name, @Surname, @Email, @Phone)
END
GO

CREATE PROCEDURE AddClientCompany @ZipCode varchar(6), @City varchar(30), @Address varchar
    (100),
                                    @companyName varchar(100), @nip nvarchar(15), @phone varchar
    (20), @email varchar(100)
AS
BEGIN
    SET NOCOUNT ON;
    INSERT INTO Clients(zip_code, city, address)
    VALUES (@ZipCode, @City, @Address)

    INSERT INTO Companies (companyName, nip, phone, clients_id, email)
    VALUES (@companyName, @nip, @phone, scope_identity(), @email)
END
GO

-- Add participant
CREATE PROCEDURE AddParticipant @Name varchar(100), @Surname varchar(100),
                                @Email varchar(100), @Phone varchar(20)
AS
BEGIN
    SET NOCOUNT ON;
    INSERT INTO Participants(name, surname, email, phone)
    VALUES (@Name, @Surname, @Email, @Phone)
END
GO

CREATE PROCEDURE AddPayment @ReservationID int, @Value money
AS
BEGIN
    SET NOCOUNT ON;
    INSERT INTO Payments(reservation_id, value)
    VALUES (@ReservationID, @Value) -- date = getDate() by default
END
GO

CREATE PROCEDURE AddConference @Name varchar(100),
                               @description text = null
AS
BEGIN
    SET NOCOUNT ON;
    INSERT INTO Conferences(name, description)
    VALUES (@Name, @description)
END
GO

CREATE PROCEDURE AddConferenceDay @ConferenceId int,
                                  @Date date,
                                  @StandardPrice money,
                                  @StudentDiscount int,
                                  @NumberOfSeats int
AS
BEGIN
    SET NOCOUNT ON;
    INSERT INTO Conference_days
    (conference_id, date, standard_price, student_discount, number_of_seats)
    VALUES (@ConferenceId, @Date, @StandardPrice, @StudentDiscount, @NumberOfSeats)
END
```

```sql
GO

CREATE PROCEDURE AddConferenceWithDays @Name varchar(100),
                                       @numberOfDays int,
                                       @startDate date,
                                       @price money,
                                       @studentDiscount decimal(4, 4),
                                       @numberOfSeats int,
                                       @description text = null
AS
BEGIN
    SET NOCOUNT ON;
    INSERT INTO Conferences(name, description)
    VALUES (@Name, @description)

    DECLARE @confId int = SCOPE_IDENTITY()

    WHILE @numberOfDays > 0
        BEGIN
            EXEC AddConferenceDay
                 @confId,
                 @startDate,
                 @price,
                 @studentDiscount,
                 @numberOfSeats;
            SET @startDate = DATEADD(DAY, 1, @startDate)
            SET @numberOfSeats = @numberOfSeats - 1
        END
END
GO

CREATE PROCEDURE AddEarlySignupDiscount @ConferenceDayId int,
                                        @EndDate datetime,
                                        @Discount decimal(4, 4)
AS
BEGIN
    SET NOCOUNT ON;
    INSERT INTO Early_signup_discounts(conference_day_id, end_date, discount)
    VALUES (@ConferenceDayId, @EndDate, @Discount)
END
GO

CREATE PROCEDURE AddEarlySignupDiscountToAllInConf @ConferenceId int,
                                                   @EndDate datetime,
                                                   @Discount decimal(4, 4)
AS
BEGIN
    SET NOCOUNT ON;
    CREATE TABLE #ids
    (
        rn int,
        id int
    )
    INSERT INTO #ids
    SELECT DISTINCT row_number() over (order by conference_day_id) as rn, conference_day_id as
     id
    FROM Conference_days
    WHERE conference_id = @ConferenceId;

    DECLARE @id int
    DECLARE @totalrows int = (select count(*) from #ids)
    DECLARE @currentrow int = 1

    WHILE @currentrow <= @totalrows
        BEGIN
            set @id = (select id from #ids where rn = @currentrow)
            exec AddEarlySignupDiscount
                 @id,
                 @EndDate,
                 @Discount
            set @currentrow = @currentrow + 1
        END
END
GO

CREATE PROCEDURE AddWorkshop @conference_day_id int,
                                    @start_time datetime,
                                    @end_time datetime,
                                    @topic varchar(100),
                                    @price money,
                                    @number_of_seats int
AS
BEGIN
    SET NOCOUNT ON;
    INSERT INTO Workshops(conference_day_id, start_time, end_time, topic, price,
    number_of_seats)
    VALUES (@conference_day_id, @start_time, @end_time, @topic, @price, @number_of_seats)
END
GO


CREATE PROCEDURE RegisterParticipantForConferenceDay @reservation_id int, @Participant_id int,
     @is_student bit
AS
BEGIN
    SET NOCOUNT ON;
    INSERT INTO Conference_day_registration(reservation_id, Participant_id, is_student)
    VALUES (@reservation_id, @Participant_id, @is_student)
end
go

CREATE PROCEDURE RegisterParticipantForWorkshop @reservation_id int, @Participant_id int
AS
BEGIN
    SET NOCOUNT ON;
    INSERT INTO Workshop_registration(reservation_id, Participant_id)
    VALUES (@reservation_id, @Participant_id)
end
go

CREATE PROCEDURE AddConferenceDayReservation @conference_day_id int,
                                             @clients_id int,
                                             @reservation_date datetime,
                                             @due_price datetime,
                                             @adult_seats int,
                                             @student_seats int
AS
BEGIN
    SET NOCOUNT ON;
    IF ((SELECT COUNT(conference_day_id) FROM Conference_days WHERE conference_day_id =
    @conference_day_id) = 0)
        BEGIN
            THROW 52000,'There is no such conference day in database', 1;
        END
    INSERT INTO Conference_day_reservations(conference_day_id, clients_id, reservation_date,
    due_price,
                                            adult_seats, student_seats)
    VALUES (@conference_day_id, @clients_id, @reservation_date, @due_price, @adult_seats,
    @student_seats)
end
go

CREATE PROCEDURE AddConferenceReservation @conference_id int,
                                          @clients_id int,
                                          @reservation_date datetime,
                                          @due_price datetime,
                                          @adult_seats int,
                                          @student_seats int
AS
BEGIN
    SET NOCOUNT ON;
    IF ((SELECT COUNT(conference_id) FROM Conferences WHERE Conference_id = @conference_id) =
    0)
        BEGIN
            THROW 52000,'There is no such conference in database', 1;
        END

    CREATE TABLE #ids
    (
        rn int,
        id int
    )
    INSERT INTO #ids
    SELECT DISTINCT row_number() over (order by conference_day_id) as rn, conference_day_id as
     id
    FROM Conference_days
    WHERE conference_id = @conference_id;


    DECLARE @id int
    DECLARE @totalrows int = (select count(*) from #ids)
    IF @totalrows=0
        BEGIN
            THROW 52000,'This conference has no days defined', 1;
        END
    DECLARE @currentrow int = 1

    WHILE @currentrow <= @totalrows
        BEGIN
            SET @id = (select id from #ids where rn = @currentrow)
            EXEC AddConferenceDayReservation
                 @id,
                 @clients_id,
                 @reservation_date,
                 @due_price,
                 @adult_seats,
                 @student_seats
            SET @currentrow = @currentrow + 1
        END

END
GO

CREATE PROCEDURE AddWorkshopReservation @workshop_id int,
                                        @conf_reservation_id int,
                                        @nr_of_seats int,
                                        @weeks_to_pay int
AS
BEGIN
    SET NOCOUNT ON;
    IF ((SELECT COUNT(workshop_id) FROM Workshops WHERE workshop_id = @workshop_id) = 0)
        BEGIN
            THROW 52000,'There is no such workshop in database', 1;
        END

    INSERT INTO Workshop_reservations(workshop_id, Conference_day_res_id, nr_of_seats,
    due_price)
    VALUES (@workshop_id, @conf_reservation_id, @nr_of_seats, DATEADD(week, @weeks_to_pay,
    GETDATE()))
END
GO
```

../Procedures/AdditionProcedures.sql

Procedury usuwające dane z bazy:

```
CREATE PROCEDURE DeleteWorkshopRegistration @ParticipantID int, @ReservationID int
AS
BEGIN
    SET NOCOUNT ON;
    BEGIN TRY
        BEGIN TRAN DELETE
                    FROM Workshop_registration
                    WHERE reservation_id = @ReservationId
                      AND Participant_id = @ParticipantID
        COMMIT TRAN
    END TRY
    BEGIN CATCH
        PRINT error_message() ROLLBACK TRANSACTION
    END CATCH
END
GO

CREATE PROCEDURE DeleteconferenceRegistration @ParticipantID int, @ReservationID int
AS
BEGIN
    SET NOCOUNT ON;
    BEGIN TRY
        BEGIN TRAN DELETE
```

```
                    FROM Conference_day_registration
                    WHERE reservation_id = @ReservationId
                      AND Participant_id = @ParticipantID
        COMMIT TRAN
    END TRY
    BEGIN CATCH
        PRINT error_message() ROLLBACK TRANSACTION
    END CATCH
END
GO
```

../Procedures/DeleteProcedures.sql

Procedury aktualizujące informacje zawarte w bazie danych:

```
-- procedura pozwalaj ca zmniejszy  ilo     zarezerwowanych miejsc na warsztaty, pod
    warunkiem  e  na dan  rezerwacj
-- nie zarejestrowa o si  ju  wi cej os b
CREATE PROCEDURE DecreaseNumberOfBookedPlacesForWorkshop @reservation_id int, @new_nr_of_seats
     int
AS
BEGIN
    SET NOCOUNT ON;
    DECLARE @already_registered_participants int = (SELECT COUNT(*)
                                                    FROM Workshop_registration
                                                    WHERE reservation_id = @reservation_id)
    IF @new_nr_of_seats < (SELECT nr_of_seats FROM Workshop_reservations WHERE reservation_id
    = @reservation_id)
        BEGIN
            IF (@already_registered_participants <= @new_nr_of_seats)
                BEGIN
                    UPDATE Workshop_reservations
                    SET nr_of_seats = @new_nr_of_seats
                    WHERE reservation_id = @reservation_id
                    PRINT 'success'
                END
            ELSE
                BEGIN
                    PRINT 'cannot decrease number of booked places, as ' +
                          CAST(@already_registered_participants AS varchar(4)) +
                          ' participants already registered for the workshop on this
    reservation'
                END
        END
END
GO

-- procedura dezaktywuj ca rezerwacj  warsztatu
CREATE PROCEDURE DeactivateWorkshopReservation @reservation_id int
AS
BEGIN
    SET NOCOUNT ON;
    DECLARE @currentlyActive bit = (SELECT active FROM Workshop_reservations WHERE
    reservation_id = @reservation_id)
    IF @currentlyActive = 1
        BEGIN
            UPDATE Workshop_reservations
            SET active = 0
            WHERE reservation_id = @reservation_id
        END
END
GO

-- procedura dezaktywuj ca rezerwacj  konferencji
CREATE PROCEDURE DeactivateConferenceReservation @reservation_id int
AS
BEGIN
    SET NOCOUNT ON;
    DECLARE @currentlyActive bit = (SELECT active FROM Conference_day_reservations WHERE
    reservation_id = @reservation_id)
    IF @currentlyActive = 1
        BEGIN
            UPDATE Conference_day_reservations
            SET active = 0
            WHERE reservation_id = @reservation_id
        END
END
```

```
GO
```

../Procedures/UpdateProcedures.sql

Triggery

```
-- ============================================
-- Triggery dla tabeli Conference_days
-- ============================================
-- sprawdzenie czy na dan  konferencje nie zosta y dodane dwa te same dni konferencji
CREATE TRIGGER CheckForTwoTheSameConferenceDays
    ON Conference_days
    AFTER INSERT, UPDATE AS
BEGIN
    DECLARE @Date date = (SELECT date FROM inserted)
    DECLARE @ConferenceId int = (SELECT conference_id FROM inserted)
    IF ((SELECT COUNT(conference_day_id)
         FROM Conference_days
         WHERE (date = @Date)
           AND (conference_id = @ConferenceId)) > 1)
        BEGIN
            DECLARE @message varchar(100) = 'Day ' + CAST(@Date as varchar(11)) +
                                            ' has already been added for this conference';
            THROW 52000,@message,1 ROLLBACK TRANSACTION
        END
END
-- ============================================
-- Triggery dla tabeli Conference_day_reservations
-- ============================================
-- przy rzerwacji miejsc na konferencje sprawdza czy jest odpowiednia ilo    wolnych miejsc
CREATE TRIGGER CheckIfEnoughSeatsAvailableForConference
    ON Conference_day_reservations
    AFTER INSERT, UPDATE AS
BEGIN
    DECLARE @ConferenceDayID int = (SELECT conference_day_id FROM inserted)
    DECLARE @RequestedSeats int = (SELECT (adult_seats + student_seats) FROM inserted)
    DECLARE @FreeSeats int = (SELECT [dbo].[conferenceFreeSeats](@ConferenceDayID))
    IF (@FreeSeats < @RequestedSeats)
        BEGIN
            DECLARE @Message varchar(100) = 'There are only ' + CAST(@FreeSeats as varchar(10)
    ) +
                                            ' places left for this conference day.';
            THROW 52000,@Message,1 ROLLBACK TRANSACTION
        END
END

-- ============================================
-- Triggery dla tabeli Conference_day_registration
-- ============================================
-- trigger zapewniaj cy aby na dan  rezerwacj  dnia konferencji nie zapisa o si  wi cej
    os b
-- ni  jest zarezerwowanych miejsc
CREATE TRIGGER CheckIfConfReservationNotFull
    ON Conference_day_registration
    AFTER INSERT, UPDATE AS
BEGIN
    DECLARE @reservationId int = (SELECT reservation_id FROM inserted)
    DECLARE @isStudent bit = (SELECT is_student FROM inserted)
    DECLARE @seatsTaken int = (SELECT COUNT(participant_id)
                               FROM Conference_day_registration
                               WHERE reservation_id = @reservationId
                                 AND is_student = @isStudent)
    DECLARE @seatsFree int = (SELECT IIF(@isStudent = 1, student_seats, adult_seats)
                              FROM Conference_day_reservations
                              WHERE reservation_id = @reservationId)
    IF (@seatsTaken > @seatsFree)
        BEGIN
            DECLARE @message varchar(100) = 'nr of participants for this reservation is
    exceeded, ' +
                                            CAST(@seatsTaken as varchar(3)) +
                                            ' participants are already registered';
            THROW 52000,@message,1 ROLLBACK TRANSACTION
        END
END

-- sprawdzenie czy uczestnik zapisuje si  na aktywn  rezerwacj
CREATE TRIGGER CheckIfCReservationActive
    ON Conference_day_registration
    AFTER INSERT, UPDATE AS
BEGIN
```

```
    DECLARE @ReservationId int = (SELECT reservation_id FROM inserted)
    IF ((SELECT active FROM Conference_day_reservations WHERE reservation_id = @ReservationId)
        = 0)
        BEGIN
            DECLARE @message varchar(100) = 'corresponding reservation is not active';
            THROW 52000,@message,1 ROLLBACK TRANSACTION
        END
END
```

../Triggers/Conferences.sql

```
-- ========================================
-- Triggery dla tabeli Workshops
-- ========================================
-- sprawdzenie czy godzina rozpocz cia warsztat w jest wcze niejsza ni  godzina jego
    zako czenia
CREATE TRIGGER StartHourLtEndHour
    ON Workshops
    AFTER INSERT, UPDATE AS
BEGIN
    DECLARE @StartTime date = (SELECT start_time FROM inserted)
    DECLARE @EndTime date = (SELECT end_time FROM inserted)
    IF (@StartTime > @EndTime)
        BEGIN
            DECLARE @message varchar(100) = 'Workshop cant begin after it ends.';
            THROW 52000,@message,1 ROLLBACK TRANSACTION
        END
END

-- sprawdzenie czy tworzone warsztaty nie maj  wi kszej ilo ci miejsc ni  to ogranicza
    odpowiadaj cy dzie  konferencji
CREATE TRIGGER WorkshopPlacesLtConfPlaces
    ON Workshops
    AFTER INSERT, UPDATE AS
BEGIN
    DECLARE @confDayId int = (SELECT conference_day_id FROM inserted)
    DECLARE @nrOfseatsW int = (SELECT number_of_seats FROM inserted)
    DECLARE @nrOfseatsC int = (SELECT number_of_seats FROM Conference_days WHERE
    conference_day_id = @confDayId)
    IF (@nrOfseatsW > @nrOfseatsC)
        BEGIN
            DECLARE @message varchar(100) = 'Conference day offers less seats than proposed
    workshop';
            THROW 52000,@message,1 ROLLBACK TRANSACTION
        END
END

-- ========================================
-- Triggery dla tabeli Workshop_reservations
-- ========================================
-- przy rzerwacji miejsc na warsztaty sprawdza czy jest odpowiednia ilo   wolnych miejsc na
    warsztatach
CREATE TRIGGER CheckIfEnoughSeatsAvailableForWorkshop
    ON Workshop_reservations
    AFTER INSERT, UPDATE AS
BEGIN
    DECLARE @WorkshopID int = (SELECT workshop_id FROM inserted)
    DECLARE @RequestedSeats int = (SELECT nr_of_seats FROM inserted)
    DECLARE @FreeSeats int = (SELECT [dbo].[workshopFreeSeats](@WorkshopID))
    IF (@FreeSeats < @RequestedSeats)
        BEGIN
            DECLARE @Message varchar(100) = 'There are only ' + CAST(@FreeSeats as varchar(10)
    ) +
                                            ' places left for this conference day.';
            THROW 52000,@Message,1 ROLLBACK TRANSACTION
        END
END


-- sprawdzenie czy klient nie pr buje zarezerwowa  warsztatu odbywaj cego si  w inny
    dzie
-- ni  powi zana z t  rezerwacj  rezerwacja dnia konferencji
CREATE TRIGGER CheckIfWorkshopAndConfOnSameDay
    ON Workshop_reservations
    AFTER INSERT, UPDATE AS
BEGIN
```

```
    DECLARE @WorkshopID int = (SELECT workshop_id FROM inserted)
    DECLARE @WorkshopDate date = (SELECT CONVERT(date, start_time)
                                    FROM Workshops
                                    WHERE Workshops.workshop_id = @WorkshopID)
    DECLARE @ConfDayResID int = (SELECT Conference_day_res_id FROM inserted)
    DECLARE @ConfDayID int = (SELECT conference_day_id
                              FROM Conference_day_reservations
                              WHERE reservation_id = @ConfDayResID)
    DECLARE @ConfDayDate date = (SELECT date FROM Conference_days WHERE conference_day_id =
    @ConfDayID)
    IF (@ConfDayDate <> @WorkshopDate)
        BEGIN
            DECLARE @Message varchar(100) = 'workshop date differs from the associated
    conference day date';
            THROW 52000,@Message,1 ROLLBACK TRANSACTION
        END
END

-- sprawdzenie czy odpowiadaj ca rezerwacji warsztatu rezerwacja dnia konferencji jest
    aktywna
CREATE TRIGGER CheckIfCorrespConfReservationActive
    ON Workshop_reservations
    AFTER INSERT, UPDATE AS
BEGIN
    DECLARE @ConfResID int = (SELECT Conference_day_res_id FROM inserted)
    IF ((SELECT active FROM Conference_day_reservations cdr WHERE cdr.reservation_id =
    @ConfResID) = 0)
        BEGIN
            DECLARE @Message varchar(100) = 'Corresponding conference day reservation is not
    active';
            THROW 52000,@Message,1 ROLLBACK TRANSACTION
        END
END

-- procedura usuwaj ca uczestnik w warsztat w zapisanych na dezaktywowan  rezerwacj
CREATE TRIGGER OnDeactivateWReservation
    ON Workshop_reservations
    AFTER UPDATE AS
BEGIN
    DECLARE @active bit = (SELECT active FROM inserted)
    DECLARE @ReservationID int = (SELECT reservation_id FROM inserted)
    IF (@active = 0)
        BEGIN TRY
        BEGIN TRAN DELETE
                   FROM Workshop_registration
                   WHERE reservation_id = @ReservationId
        COMMIT TRAN
    END TRY
    BEGIN CATCH
        PRINT error_message() ROLLBACK TRANSACTION
    END CATCH
END
-- ===================================
-- Triggery dla tabeli Workshop_registration
-- ===================================
-- sprawdzenie czy uczestnik zapisuj cy si  na warsztaty jest ju  zapisany na odpowiedni
    dzie  konferencji
CREATE TRIGGER CheckIfRegisteredForTheDay
    ON Workshop_registration
    AFTER INSERT, UPDATE AS
BEGIN
    DECLARE @ReservationId int = (SELECT reservation_id FROM inserted)
    DECLARE @ParticipantID int = (SELECT participant_id FROM inserted)
    DECLARE @ConferenceDayID int = (SELECT conference_day_id
                                    FROM Workshops
                                             INNER JOIN Workshop_reservations
                                                        on Workshops.workshop_id =
    Workshop_reservations.workshop_id
                                    WHERE Workshop_reservations.reservation_id =
    @ReservationId)
    IF (@ParticipantID not in (SELECT participant_id
                               FROM Conference_day_registration Cdr
                                        INNER JOIN Conference_day_reservations C on Cdr.
    reservation_id = C.reservation_id
                               WHERE C.conference_day_id = @ConferenceDayID))
        BEGIN
```

```
            DECLARE @message varchar(100) = 'Participant nr ' + CAST(@ParticipantID as varchar(3)) +
                                        ' is not registered for the corresponding conference day';
            THROW 52000,@message,1 ROLLBACK TRANSACTION
        END
END

-- sprawdzenie czy uczestnik zapisuj cy si  na warsztaty nie jest ju  zapisany na inne warsztaty
-- odbywaj ce si  w tym samym czasie
CREATE TRIGGER CheckIfNotRegisteredForOtherWorkshop
    ON Workshop_registration
    AFTER INSERT, UPDATE AS
BEGIN
    DECLARE @ReservationId int = (SELECT reservation_id FROM inserted)
    DECLARE @ParticipantID int = (SELECT participant_id FROM inserted)
    DECLARE @StartTime datetime = (SELECT start_time
                                FROM Workshops
                                        INNER JOIN Workshop_reservations Wr on Workshops.workshop_id = Wr.workshop_id
                                WHERE Wr.reservation_id = @ReservationId)
    DECLARE @EndTime datetime = (SELECT end_time
                              FROM Workshops
                                      INNER JOIN Workshop_reservations Wr on Workshops.workshop_id = Wr.workshop_id
                              WHERE Wr.reservation_id = @ReservationId)
    DECLARE @CollidingWorkshops int = (SELECT Count(W.workshop_id)
                                    FROM Workshop_reservations Wrs
                                            INNER JOIN Workshop_registration Wrg
                                                      on Wrs.reservation_id = Wrg.reservation_id AND
                                                         Wrg.Participant_id = @ParticipantID
                                            INNER JOIN Workshops W on Wrs.workshop_id = W.workshop_id
                                    WHERE ((W.start_time BETWEEN @StartTime AND @EndTime)
                                        OR (W.end_time BETWEEN @StartTime AND @EndTime)))
    IF (@CollidingWorkshops
        > 1) -- already registered for the day
        BEGIN
            DECLARE @message varchar(100) = 'Participant nr ' + CAST(@ParticipantID as varchar(3)) +
                                        ' already registered for ' + CAST(@CollidingWorkshops as varchar(3)) +
                                        ' workshops at this time';
            THROW 52000,@message,1 ROLLBACK TRANSACTION
        END
END

-- trigger zapewniaj cy aby na dan  rezerwacj  warsztat w nie zapisa o si  wi cej os b
-- ni  jest zarezerwowanych miejsc
CREATE TRIGGER CheckIfWorkshopReservationNotFull
    ON Workshop_registration
    AFTER INSERT, UPDATE AS
BEGIN
    DECLARE @ReservationId int = (SELECT reservation_id FROM inserted)
    DECLARE @PartForReserv int = (SELECT COUNT(Participant_id)
                               FROM Workshop_registration Wr
                               WHERE Wr.reservation_id = @ReservationId)
    DECLARE @ReservedSeats int = (SELECT nr_of_seats FROM Workshop_reservations WHERE reservation_id = @ReservationId)
    IF (@PartForReserv > @ReservedSeats)
        BEGIN
            DECLARE @message varchar(100) = 'nr of participants for this reservation is exceeded, ' +
                                        CAST(@PartForReserv as varchar(3)) +
                                        ' participants are already registered';
            THROW 52000,@message,1 ROLLBACK TRANSACTION
        END
END

-- sprawdzenie czy uczestnik zapisuje si  na aktywn  rezerwacj
CREATE TRIGGER CheckIfWReservationActive
    ON Workshop_registration
    AFTER INSERT, UPDATE AS
```

```sql
BEGIN
    DECLARE @ReservationId int = (SELECT reservation_id FROM inserted)
    IF ((SELECT active FROM Workshop_reservations WHERE reservation_id = @ReservationId)
        = 0)
        BEGIN
            DECLARE @message varchar(100) = 'corresponding reservation is not active';
            THROW 52000,@message,1 ROLLBACK TRANSACTION
        END
END
```

../Triggers/Workshops.sql

Generator danych

```python
from ParticipantsGenerator import *
from ClientsGenerator import *
from ConfDayResGenerator import *
from ConfDaysGenerator import *
from EsdGenerator import *
from ConferenceGenerator import *
from WorkshopGenerator import *
from WorkshopResGen import *
from ConfRegistrationGen import *
from WorkshopRegistrationGen import *
from PaymentGen import *

from random import Random
from faker import Faker

faker = Faker(['pl_PL'])
rand = Random()

generators = []

part_gen = ParticipantsGenerator(rand)
part_gen.make(None, 4000)
#part_gen.make(None, 5)

clients_gen = ClientsGenerator(part_gen)
generators.append(clients_gen)
clients_gen.make(400)
#clients_gen.make(10)

conf_gen = ConferenceGenerator(rand, faker)
generators.append(conf_gen)
conf_gen.make(72)
#conf_gen.make(3)

day_gen = ConfDaysGenerator(rand, faker)
generators.append(day_gen)
day_gen.make(conf_gen.conferences)

esd_gen = EsdGenerator(rand, faker)
generators.append(esd_gen)
esd_gen.make(day_gen.days)

day_res_gen = ConfDayResGenerator(clients_gen, rand, faker)
generators.append(day_res_gen)
day_res_gen.make(day_gen.days)

wor_gen = WorkshopGenerator(part_gen, rand, faker)
generators.append(wor_gen)
wor_gen.make(day_gen.days)

wor_res_gen = WorkshopResGen(day_res_gen)
generators.append(wor_res_gen)
wor_res_gen.make(wor_gen.workshops)

conf_reg_gen = ConfRegistrationGen(rand, part_gen)
generators.append(conf_reg_gen)
conf_reg_gen.make(day_res_gen.reservations)

wor_reg_gen = WorkshopRegistrationGen(rand, part_gen)
generators.append(wor_reg_gen)
wor_reg_gen.make(wor_res_gen.reservations)

pay_gen = PaymentGen(rand, faker)
generators.append(pay_gen)
pay_gen.make(day_res_gen.reservations)

for g in generators:
    print(g.to_sql())
```

../Generator/main.py

```python
from datetime import datetime, timedelta, time
```

```

class AbstractClass:
    def random_time(self, date):
        return datetime.combine(date, time(self.rand.randint(0, 23), self.rand.randint(0, 59))
    )
```

../Generator/AbstractClass.py

```
def table_to_sql(table, n_row=True):
    res = '\n' if n_row else ''
    res += table[0].to_sql()
    lid = 0
    for v in range(1, len(table)):
        if v - lid < 950:
            res += ','
            res += table[v].to_sql(False)
        else:
            res += '\n'
            res += table[v].to_sql()
            lid = v
    return res
```

../Generator/AbstractGenerator.py

```
class Client:
    def __init__(self, cl_id, faker):
        self.cl_id = cl_id
        self.faker = faker

        add = self.faker.address().split('\n')
        self.address = add[0]
        self.zip_code = add[-1].split(' ')[0]
        self.city = ' '.join(add[-1].split(' ')[1:])

    def to_sql(self, start=True):
        values = "(" + str(self.cl_id) + ",\'" + self.zip_code + "\',\'" + self.city + "\',\'"
     + self.address + "\')"
        return "INSERT INTO CLIENTS (id,zip_code, city, address) " \
                "VALUES "+ values if start else values
```

../Generator/Client.py

```
from faker import Faker
import random
from ParticipantsGenerator import *
from Client import *
from Company import *


def table_to_sql(table):
    res = '\n'
    res += table[0].to_sql()
    lid = 0
    for v in range(1, len(table)):
        if v - lid < 950:
            res += ','
            res += table[v].to_sql(False)
        else:
            lid = v
            res += '\n'
            res += table[v].to_sql()
    return res


class ClientsGenerator:
    def __init__(self, participants_gen, next_client_id=1):
        self.faker = Faker(['pl_PL'])
        self.rand = random.Random()

        self.next_client_id = next_client_id
        self.participants_gen = participants_gen
        self.clients = []
        self.companies = []

    def choice(self):
        return self.rand.choice(self.clients)

    def clients_count(self):
        return len(self.clients)

    def make(self, n=1):
        for _ in range(n):
            cl = Client(self.next_client_id, self.faker)
            if self.rand.randint(0, 1) == 0:
                cm = Company(self.next_client_id, self.faker, self.rand)
                self.companies.append(cm)
            else:
                self.participants_gen.make(self.next_client_id)
            self.next_client_id += 1
            self.clients.append(cl)

    def to_sql(self):
        res = 'SET IDENTITY_INSERT Clients ON'
        res += table_to_sql(self.clients)

        res += '\nSET IDENTITY_INSERT Clients OFF'
        res += table_to_sql(self.companies)
        res += '\n'
        res += self.participants_gen.to_sql()

        self.clients = []
        self.companies = []
        return res
```

../Generator/ClientsGenerator.py

```
class Company:
    def __init__(self, clients_id, faker, rand):
        self.faker = faker
        self.rand = rand
        self.clients_id = clients_id

        self.name = self.faker.company()
        self.phone = self.faker.phone_number()
        self.email = self.faker.email()
        self.nip = self.random_nip()

    def random_nip(self):
        res = ''
        sum = 0
        weights = [6, 5, 7, 2, 3, 4, 5, 6, 7]
        for i in range(8):
            k = self.rand.randint(1 if i < 3 else 0, 9)
            sum += weights[i] * k
            res += str(k)
        k = self.rand.randint(0, 9)
        if (sum + (k * weights[8])) % 11 == 10:
            k += (1 if k + 1 < 10 else -1)
        res += str(k)
        sum += k * weights[8]
        res += str(sum % 11)
        return res

    def to_sql(self, start=True):
        values = "(\'" + self.name + "\',\'" + self.nip + "\',\'" + self.phone + "\'," + str(
            self.clients_id) + ",\'" + self.email + "\')"
        return "INSERT INTO COMPANIES (companyName, nip, phone, clients_id, email) VALUES "+
    values if start else values
```

../Generator/Company.py

```
from AbstractClass import *


class ConfDayReservation(AbstractClass):
    def __init__(self, res_id, clients_id, day, part_count, faker, rand):
        self.faker = faker
        self.rand = rand

        self.res_id = res_id
        self.day_id = day.day_id
```

```python
        self.clients_id = clients_id

        self.date = self.random_time(self.faker.date_between(start_date=datetime.today(), end_date=day.date))
        self.active = 1 if self.rand.randint(1, 1000) % 8 == 0 else 0
        self.due_price = self.date + timedelta(weeks=self.rand.randint(1, 4))
        self.adult_seats = self.rand.randint(1, min(day.free_seats, part_count))
        self.student_seats = self.rand.randint(0, max(0, min(day.free_seats, part_count) - self.adult_seats))

        self.workshops_price = 0
        self.day_price = day.price * self.adult_seats
        self.day_price += day.price * (1 - day.stud_disc) * self.student_seats
        esds = list(filter(lambda x: x.date > self.date.date(), day.esds))
        esds = sorted(esds, key=lambda x: x.date)
        esd = 0 if len(esds) == 0 else esds[0].discount
        self.day_price *= (1 - esd)

    def to_sql(self, start=True):
        values = "(" + str(
            self.res_id) + "," + str(self.day_id) + "," + str(self.clients_id) + ",\'" + str(self.date) + "\',\'" + str(
            self.active) + "\',\'" + str(self.due_price) + "\'," + str(self.adult_seats) + "," + str(
            self.student_seats) + ")"
        return "INSERT INTO Conference_day_reservations (reservation_id, conference_day_id, clients_id, reservation_date, active, due_price, adult_seats, student_seats) VALUES " + values if start else values
```

../Generator/ConfDayReservation.py

```python
from random import Random
from faker import Faker
from ConfDayReservation import *


class ConfDayResGenerator:
    def __init__(self, clients_gen, rand=Random(), faker=Faker(['pl_PL']), next_res_id=1):
        self.faker = faker
        self.rand = rand

        self.clients_gen = clients_gen
        self.next_res_id = next_res_id
        self.reservations = []

    def choice(self):
        return self.rand.choice(self.reservations)

    def res_count(self):
        return len(self.reservations)

    def to_sql(self):
        res = 'SET IDENTITY_INSERT Conference_day_reservations ON'
        res += '\n'
        res += self.reservations[0].to_sql()
        for v in range(1, len(self.reservations)):
            res += ','
            res += self.reservations[v].to_sql(False)
        res += '\nSET IDENTITY_INSERT Conference_day_reservations OFF'
        self.reservations = []
        return res

    def make(self, days):
        for day in days:
            n_res = self.rand.randint(2, self.clients_gen.clients_count() / 5)
            while day.free_seats > 0 and n_res > 0:
                n_res -= 1
                self.reservations.append(
                    ConfDayReservation(self.next_res_id, self.clients_gen.choice().cl_id, day,
                                       len(self.clients_gen.participants_gen.participants), self.faker, self.rand))
                self.next_res_id += 1
                day.free_seats -= self.reservations[-1].adult_seats
                day.free_seats -= self.reservations[-1].student_seats
```

../Generator/ConfDayResGenerator.py

```python
from faker import Faker
from random import Random
from ConferenceDay import *
from datetime import datetime, timedelta, time
from AbstractGenerator import table_to_sql


class ConfDaysGenerator:
    def __init__(self, rand=Random(), faker=Faker(['pl_PL']), next_day_id=1):
        self.faker = faker
        self.rand = rand

        self.next_day_id = next_day_id
        self.days = []

    def make(self, conferences):
        for c in conferences:
            self.days.append(ConferenceDay(self.next_day_id, c.conf_id, self.faker, self.rand)
    )
            self.next_day_id += 1
            date = self.days[-1].date
            n = self.rand.randint(2, 4)
            for _ in range(n):
                date += timedelta(days=1)
                self.days.append(ConferenceDay(self.next_day_id, c.conf_id, self.faker, self.
    rand, date))
                self.next_day_id += 1

    def to_sql(self):
        res = 'SET IDENTITY_INSERT Conference_days ON'
        res += table_to_sql(self.days)
        res += '\nSET IDENTITY_INSERT Conference_days OFF'
        self.days = []
        return res
```

../Generator/ConfDaysGenerator.py

```python
class ConferenceDay:
    def __init__(self, day_id, conf_id, faker, rand, day=None):
        self.faker = faker
        self.rand = rand

        self.day_id = day_id
        self.conf_id = conf_id
        self.price = round(self.rand.uniform(10.0, 1000.0), 2)
        self.stud_disc = round(self.rand.uniform(0.0, 0.5), 2)
        self.date = (self.faker.date_between(start_date='today', end_date='+5y')) if day is
    None else day
        self.numb_of_seats = self.rand.randint(150, 250)
        self.free_seats = self.numb_of_seats
        self.esds = []

    def to_sql(self, start=True):
        values = "(" + str(self.day_id) + "," + str(self.conf_id) + ",\'" + str(self.date) + "
    \'," + str(
            self.price) + "," + str(self.stud_disc) + ',' + str(self.numb_of_seats) + ")"
        return "INSERT INTO Conference_days (conference_day_id, conference_id, date,
    standard_price, student_discount, number_of_seats) VALUES " + values if start else values
```

../Generator/ConferenceDay.py

```python
from faker import Faker
from random import Random
from Conference import *
from AbstractGenerator import table_to_sql


class ConferenceGenerator:
    def __init__(self, rand=Random(), faker=Faker(['pl_PL']), next_conference_id=1):
        self.faker = faker
        self.rand = rand

        self.next_conference_id = next_conference_id
        self.conferences = []
    def to_sql(self):
        res = 'SET IDENTITY_INSERT Conferences ON'
        res += table_to_sql(self.conferences)
        res += '\nSET IDENTITY_INSERT Conferences OFF'
        self.conferences = []
        return res

    def make(self, n=1):
        for _ in range(n):
            self.conferences.append(Conference(self.next_conference_id, self.faker))
            self.next_conference_id += 1
```

../Generator/ConferenceGenerator.py

```
class Conference:
    def __init__(self, conf_id, faker):
        self.faker = faker

        self.conf_id = conf_id
        self.name = self.faker.bs()
        self.description = self.faker.text()

    def to_sql(self, start=True):
        values = "(" + str(self.conf_id) + ",\'" + self.name + '\',\'' + self.description + "
    \')"
        return "INSERT INTO Conferences (Conference_id, name, description) VALUES " + values
    if start else values
```

../Generator/Conference.py

```
from ConfRegistration import *
from AbstractGenerator import *


class ConfRegistrationGen:
    def __init__(self, rand, part_gen):
        self.rand = rand

        self.part_gen = part_gen
        self.registrations = []

    def to_sql(self):
        res = table_to_sql(self.registrations, False)
        self.registrations = []
        return res

    def make(self, reservations):
        for res in reservations:
            if res.active == 0:
                continue
            parts = set([p.part_id for p in self.part_gen.participants])

            for _ in range(res.adult_seats):
                p = self.rand.sample(parts, 1)
                self.registrations.append(ConfRegistration(res.res_id, p[0], 1, self.rand))
                parts.remove(p[0])

            for _ in range(res.student_seats):
                p = self.rand.sample(parts, 1)
                self.registrations.append(ConfRegistration(res.res_id, p[0], 0, self.rand))
                parts.remove(p[0])
```

../Generator/ConfRegistrationGen.py

```
class ConfRegistration:
    def __init__(self, res_id, participant_id, student, rand):
        self.rand = rand

        self.res_id = res_id
        self.student = student
        self.participant_id = participant_id

    def to_sql(self, start=True):
        values = "(" + str(self.res_id) + "," + str(self.participant_id) + "," + str(self.
    student) + ")"
```

```
        return "INSERT INTO Conference_day_registration (reservation_id, Participant_id, is_student) VALUES " + values if start else values
```
../Generator/ConfRegistration.py

```
from random import Random
from faker import Faker
from Esd import *
from AbstractGenerator import *


class EsdGenerator:
    def __init__(self, rand=Random(), faker=Faker(['pl_PL'])):
        self.faker = faker
        self.rand = rand

        self.esds = []

    def to_sql(self):
        res = table_to_sql(self.esds, False)
        self.esds = []
        return res

    def make(self, days):
        for day in days:
            for _ in range(self.rand.randint(1, 5)):
                e = Esd(self.faker, self.rand, day)
                self.esds.append(e)
                day.esds.append(e)
```
../Generator/EsdGenerator.py

```
class Esd:
    def __init__(self, faker, rand, day):
        self.faker = faker
        self.rand = rand

        self.day_id = day.day_id
        self.discount = round(self.rand.uniform(0.0, 0.5), 2)
        self.date = self.faker.date_between(start_date='-2y', end_date=day.date)

    def to_sql(self, start=True):
        values = "(" + str(self.day_id) + ",\'" + str(self.date) + "\'," + str(self.discount) + ")"
        return "INSERT INTO Early_signup_discounts (conference_day_id, end_date, discount) VALUES " + values if start else values
```
../Generator/Esd.py

```
class Participant:
    def __init__(self, part_id, faker, client_id=None):
        self.part_id = part_id
        self.client_id = client_id
        self.faker = faker

        n = self.faker.name().split(' ')
        if n[0] == 'pani' or n[0] == 'pan' or n[0] == 'Pan' or n[0] == 'Pani':
            n = n[1:]
        self.name = n[0]
        self.surname = ' '.join(n[1:])
        self.phone = self.faker.phone_number()
        self.email = self.faker.email()

    def to_sql(self, start=True):
        values = "(" + str(self.part_id) + "," + (str(
            self.client_id) if self.client_id is not None else 'null') + ",\'" + self.name + "\',\'" + self.surname + "\',\'" + self.email + "\',\'" + self.phone + "\')"
        return "INSERT INTO Participants (participant_id,clients_id, name, surname, email, phone) VALUES " + values if start else values
```
../Generator/Participant.py

```
from faker import Faker
from Participant import *
from AbstractGenerator import table_to_sql
```

```

class ParticipantsGenerator:
    def __init__(self, rand, faker=Faker(['pl_PL']), next_participant_id=1):
        self.faker = faker
        self.rand = rand

        self.next_participant_id = next_participant_id
        self.participants = []

    def choice(self):
        return self.rand.choice(self.participants)

    def part_count(self):
        return len(self.participants)

    def make(self, clients_id=None, n=1):
        for _ in range(n):
            res = Participant(self.next_participant_id, self.faker, clients_id)
            self.next_participant_id += 1
            self.participants.append(res)

    def to_sql(self):
        res = 'SET IDENTITY_INSERT Participants ON'
        res += table_to_sql(self.participants)
        res += '\nSET IDENTITY_INSERT Participants OFF'
        self.participants = []
        return res
```

../Generator/ParticipantsGenerator.py

```
from Payment import *
from AbstractGenerator import *


class PaymentGen:
    def __init__(self, rand, faker):
        self.rand = rand
        self.faker = faker
        self.payments = []

    def make(self, reservations):
        for res in reservations:
            if res.active == 0 and self.rand.randint(1,1000) % 8 != 0:
                continue
            price = res.workshops_price + res.day_price
            payed = int(price)
            if self.rand.randint(1, 1000) % 30 == 0:  # nie oplacone
                payed = 0 if self.rand.randint(0, 1000) % 2 == 0 else self.rand.randint(0, int
    (price))
            if payed == 0:
                continue
            n_payments = self.rand.randint(1, 4)
            values = [payed // n_payments for _ in range(n_payments)]
            i = 0
            while sum(values) < payed:
                values[i % n_payments] += 1
                i += 1
            i = 0
            while i < n_payments:
                p = self.rand.randint(0, values[i])
                values[i] -= p
                values[(i + 1) % n_payments] += p
                i += 1
            values[-1] += round(price - int(price), 2)
            for value in values:
                self.payments.append(Payment(value, res, self.faker, self.rand))

    def to_sql(self):
        res = table_to_sql(self.payments, False)
        self.payments = []
        return res
```

../Generator/PaymentGen.py

```
from AbstractClass import *
```

```

class Payment(AbstractClass):
    def __init__(self, value, day_res, faker, rand):
        self.faker = faker
        self.rand = rand

        self.res_id = day_res.res_id
        self.date = self.random_time(self.faker.date_between(start_date=day_res.date, end_date
    =day_res.due_price))
        self.value = value

    def to_sql(self, start=True):
        values = "(" + str(self.res_id) + ",\'" + str(self.date) + "\'," + str(self.value) + "
    )"
        return "INSERT INTO Payments (reservation_id, in_date, value) VALUES " + values if
    start else values
```

../Generator/Payment.py

```
from random import Random
from faker import Faker
from Workshop import *
from AbstractGenerator import table_to_sql


class WorkshopGenerator:
    def __init__(self, part_gen, rand=Random(), faker=Faker(['pl_PL']), next_workshop_id=1):
        self.faker = faker
        self.rand = rand

        self.part_gen = part_gen
        self.workshops = []
        self.next_workshop_id = next_workshop_id

    def to_sql(self):
        res = 'SET IDENTITY_INSERT Workshops ON'
        res += table_to_sql(self.workshops)
        res += '\nSET IDENTITY_INSERT Workshops OFF'
        self.workshops = []
        return res

    def make(self, days):
        for day in days:
            for _ in range(self.rand.randint(2, 6)):
                self.workshops.append(
                    Workshop(self.next_workshop_id, day, self.part_gen.part_count(), self.
    faker, self.rand))
                self.next_workshop_id += 1
```

../Generator/WorkshopGenerator.py

```
from datetime import datetime, timedelta, time


class Workshop:
    def __init__(self, workshop_id, day, part_count, faker, rand):
        self.faker = faker
        self.rand = rand

        self.workshop_id = workshop_id
        self.day_id = day.day_id
        self.price = round(self.rand.uniform(10.0, 1000.0), 2)
        self.start = self.random_time(day.date)
        self.end = self.random_time(day.date)
        if self.start > self.end:
            self.start, self.end = self.end, self.start
        self.topic = self.faker.bs()
        self.numb_seats = self.rand.randint(1, min(day.numb_of_seats, part_count))
        self.free_seats = self.numb_seats

    def random_time(self, date):
        return datetime.combine(date, time(self.rand.randint(0, 23), self.rand.randint(0, 59))
    )

    def to_sql(self, start=True):
        values = "(" + str(self.workshop_id) + "," + str(self.day_id) + ",\'" + str(self.start
    ) + "\',\'" + str(
            self.end) + "\',\'" + self.topic + "\'," + str(self.price) + "," + str(self.
    numb_seats) + ")"
        return "INSERT INTO Workshops (workshop_id, conference_day_id, start_time, end_time,
    topic, price, number_of_seats) VALUES " + values if start else values
```

../Generator/Workshop.py

```
from WorkshopRegistration import *
from AbstractGenerator import *


class WorkshopRegistrationGen:
    def __init__(self, rand, part_gen):
        self.rand = rand

        self.part_gen = part_gen
        self.registrations = []

    def to_sql(self):
        res = table_to_sql(self.registrations, False)
        self.registrations = []
        return res

    def make(self, reservations):
        for res in reservations:
            if res.active == 0:
                continue
            parts = set([p.part_id for p in self.part_gen.participants])
            for _ in range(res.nr_seats):
                p = self.rand.sample(parts, 1)
                self.registrations.append(WorkshopRegistration(res.res_id, p[0]))
                parts.remove(p[0])
```

../Generator/WorkshopRegistrationGen.py

```
class WorkshopRegistration:
    def __init__(self, res_id, participant_id):
        self.res_id = res_id
        self.participant_id = participant_id

    def to_sql(self, start=True):
        values = "(" + str(self.res_id) + "," + str(self.participant_id) + ")"
        return "INSERT INTO Workshop_registration (reservation_id, Participant_id) VALUES " +
    values if start else values
```

../Generator/WorkshopRegistration.py

```
from random import Random
from faker import Faker
from WorkshopRes import *
from AbstractGenerator import table_to_sql


class WorkshopResGen:
    def __init__(self, conf_day_res_gen, rand=Random(), faker=Faker(['pl_PL']), next_res_id=1)
    :
        self.faker = faker
        self.rand = rand

        self.conf_day_res_gen = conf_day_res_gen
        self.next_res_id = next_res_id
        self.reservations = []

    def to_sql(self):
        res = 'SET IDENTITY_INSERT Workshop_reservations ON'
        res += table_to_sql(self.reservations)
        res += '\nSET IDENTITY_INSERT Workshop_reservations OFF'
        self.reservations = []
        return res

    def make(self, workshops):
        for work in workshops:
            n_res = self.rand.randint(1, self.conf_day_res_gen.res_count() // 4)
            while work.free_seats > 0 and n_res > 0:
                n_res -= 1
                self.reservations.append(
                    WorkshopRes(self.next_res_id, work, self.conf_day_res_gen.choice(), self.
    faker, self.rand))
                self.next_res_id += 1
                work.free_seats -= self.reservations[-1].nr_seats
```

../Generator/WorkshopResGen.py

```
from AbstractClass import *


class WorkshopRes(AbstractClass):
    def __init__(self, res_id, workshop, conf_res, faker, rand):
        self.faker = faker
        self.rand = rand

        self.res_id = res_id
        self.work_id = workshop.workshop_id
        self.date = self.random_time(
            self.faker.date_between(start_date=datetime.today(), end_date=workshop.start.date
    ()))
        self.due_price = self.date + timedelta(weeks=self.rand.randint(1, 4))
        self.nr_seats = self.rand.randint(1, workshop.free_seats)
        self.conf_res_id = conf_res.res_id
        self.active = self.rand.randint(0, 1)

        conf_res.workshops_price += self.nr_seats * workshop.price

    def to_sql(self, start=True):
        values = "(" + str(
            self.res_id) + "," + str(self.work_id) + ",\'" + str(self.date) + "\',\'" + str(
            self.due_price) + "\'," + str(
            self.nr_seats) + ',' + str(self.conf_res_id) + ',' + str(self.active) + ")"
        return "INSERT INTO Workshop_reservations (reservation_id, workshop_id,
    reservation_date, due_price, nr_of_seats, Conference_day_res_id, active) VALUES " + values
     if start else values
```

../Generator/WorkshopRes.py

Funkcje realizowane przez system

Funkcje użytkowników:

- Administrator - Dodawanie pracowników
- Pracownik – Dodawanie organizatorów
- Organizator – Utworzenie nowej konferencji
- Organizator – Wprowadzenie informacji o kolejnych dniach konferencji
- Organizator - Wprowadzenie informacji o warsztatach
- Klient - Rejestracja w charakterze uczestnika / firmy w systemie
- Klient - Rezerwacja miejsc na konferencji
- Klient – Rezerwacja miejsc na warsztatach
- Uczestnik - Wprowadzenie danych osobowych
- Organizator - Generowanie raportów o brakujących danych osobowych
- Organizator - Generowanie identyfikatorów
- Organizator - Generowanie raportów o zarejestrowanych uczestnikach na każdy dzień i warsztat
- Organizator – Wprowadzanie progów cenowych
- Organizator – Wprowadzanie cen warsztatów
- Organizator - Generowanie informacji o płatnościach
- Organizator – Wprowadzenie informacji o wpływającej wpłacie od Klienta
- Pracownik - Generowanie danych statystycznych na temat klientów

Funkcje systemowe:

- Zapewnienie, by wszyscy zapisujący się na warsztaty spełniali wymogi (np. by byli zarejestrowani na konferencję)
- Obsługa zapisów na warsztaty (limit miejsc, inne warsztaty)
- Kalkulacja sumarycznego kosztu ponoszonego przez uczestnika w ramach udziału w konferencji
- Kalkulacja sumarycznego kosztu ponoszonego przez klienta
- Informacja o dostępnych wolnych miejscach na konferencję
- Rejestracja płatności klientów, obsługa nieopłaconych rezerwacji
- Tworzenie raportów dla organizatorów – najbardziej aktywni klienci, informacje o płatnościach klientów

Uprawnienia

1. Administrator - Całkowity dostęp do bazy.
2. Pracownik – Dostęp poziomu organizatora dla wszystkich konferencji.
3. Organizator – Dostęp do wszystkich rekordów powiązanych z konkretną konferencją. Możliwość edycji tabel warsztatów, dni konferencyjnych.
4. Klient – Rejestracja na konferencje i warsztaty, wprowadzanie danych osobowych.
5. Uczestnik – Wgląd i edycja wprowadzonych danych osobowych, podgląd warsztatów i dni konferencyjnych na jakie jest zapisany (nie może generować kosztów, jako że jest podpięty pod klienta).